Universidad Nacional de La Plata
Facultad de Informática

Patrones XML para desarrollo de aplicaciones en la Web

Trabajo de Grado de la Licenciatura en Informática

Alumno: María Laura Ponisio

Director: Prof. Dr. Gustavo Rossi

Tabla de Contenidos

Capítulo 1

Introducción

En esta tesis analizamos eXtensible Markup Language o XML desde el punto de vista de una herramienta útil para resolver problemas comunes en el desarrollo de las aplicaciones en la Web.

Investigamos XML, el nuevo integrante de la comunidad Web que se está ganando un lugar aún estando en desarrollo.

Luego de investigar también el método OOHDM, Object Oriented Hypermedia Design Method, hipermedia y patrones, desarrollamos patrones usando XML en un intento de allanar el camino hacia la incorporación definitiva de XML al conjunto de soluciones elegibles en el campo de hipermedia.

Contexto

En el mundo de las aplicaciones hipermedia existen herramientas probadas y bien establecidas . Ejemplo de esto es la ya clásica formulación de soluciones a problemas en contextos determinados mediante patrones. Sin embargo aparece ahora un nuevo integrante y como tal se está adaptando a las necesidades de la comunidad. Ese integrante es XML, quien está respaldado por el Worl Wide Web Consortium o W3C. Es justamente W3C el organismo encargado de establecer especificaciones XML.

El tema XML comprende toda una familia de especificaciones, algunas de las cuales llegaron a un estado en donde es poco probable que sufran modificaciones, por ejemplo la especificación de XML 1.0, pero el W3C aún está tabajando sobre otras. En estas condiciones, las especificaciones que establecen los diversos aspectos de XML sufren constantemente transformaciones, dejando obsoletos a los trabajos previos.

Por otra parte, XML está muy difundido y los principales desarrolladores de soft lo recomiendan y soportan generando por un lado gran interés y por otro que una parte cada vez mayor de la comunidad desee realizar su aporte. Por lo tanto constantemente aparecen nuevas propuestas, herramientas y especificaciones.

En consecuencia una de las características determinantes del contexto de esta Tesis es la gran velocidad de cambio y por lo tanto la obsolescencia de las especificaciones. Al desarrollar este trabajo tomamos en cuenta el conjunto de

especificaciones, propuestas y programas procesadores relacionados a XML en el estado en que se encuentran a noviembre de 2000.

Objetivo

El objetivo de este trabajo es desarrollar una especificación mediante patrones de cómo definir distintas visiones de la información en un sistema utilizando la familia XML, y establecidos principios de la investigación en hypermedia.

La especificación consiste en desarrollar patrones a usar teniendo en cuenta los diferentes contextos y problemas de las aplicaciones y la particularidad que tienen las mismas de necesitar mostrar la información de diversas formas.

No hay que pretender ver los patrones en el sentido estricto tradicional del clásico libro de Gamma, Design Patterns Elements of Reusable Object-Oriented Software [Gamma+95], sino como problemas y soluciones que aparecen al visualizar la información en un sistema con XML. Tomamos la libertad de usar la palabra pattern y darle a la especificación formato de pattern porque es cómodo, práctico y entendendible definirla así. Las ventajas y utilidad de trabajar con patterns y con formato de pattern serán evidentes al leer el capítulo 2.

Además, como el objetivo es también didáctico, incluímos una buena cantidad de ejemplos de uso de XML.

Motivación

Cada vez más se habla y usa XML y la familia de recomendaciones asociada. Por ser un nuevo participante de la comunidad Web, XML recién está ocupando su lugar en los diseños e implementación de aplicaciones.

Todavía tiene que vencer los prejuicios y la desinformación que llevan a desecharlo sin considerarlo seriamente como parte de la solución.

Su novedad hace que existan pocos ejemplos prácticos. En consecuencia muchos de los ejemplos son los que ofrece el organismo que administra las especificaciones XML, el Worl Wide Web Consortium (W3C) o de los fabricantes de herramientas para XML.

No encontramos aún una clasificación importante de soluciones probadas a problemas repetidos en contextos determinados. Esto sugiere entonces la necesidad una guía que ayude a saber en qué casos y cómo aplicar XML con propiedad. Tener un catálogo de patrones XML para distintos tipos de vistas puede ser útil y práctico al momento de decidir cómo y cuándo aplicar XML.

Por otra parte el tema XML es muy amplio, porque abarca un conjunto de especificaciones que crece día a día. En consecuencia, al enfrentarse a tanta información, los diseñadores y programadores pueden caer en la tentación de desechar la idea de usar XML y preferir usar herramientas quizás menos adecuadas para resolver el problema, pero conocidas. Otro riesgo que corren es el de utilizar sólo una parte del conjunto de especificaciones XML, obteniendo como resultado, al forzar la herramienta usando el miembro de la familia XML equivocado, malas experiencias y desarrollos pobres e innecesariamente complicados.

Es interesante entonces investigar los casos en los que XML ofrece una solución eficaz, económica e integrada.

La falta de información sobre XML, las excelentes posibilidades de integración y la capacidad propia de XML de separar los datos de la presentación son también motivos que nos impulsan a investigar este tema. Como resultado de la investigación deseamos aportar soluciones concretas, fáciles y rápidas de entender. Por lo tanto elegimos el formato de patrones para plantear soluciones específicas.

Finalmente, el tema de personalizar la manera de presentar la información es muy significativo en el ambiente de aplicaciones hipermedia en la Web, en donde el usuario tiene el rol principal. Queremos entonces obtener distintas vistas de la información en aplicaciones hypermedia donde la existencia de diversas visiones de la información sea fundamental.

Hay mucho por hacer, sin empargo esperamos que este trabajo abra un camino para usar XML en el desarrollo de aplicaciones hipermedia en la Web.

Organización del trabajo

Comenzamos con un estudio de hipermedia y patrones para conocer las herramientas ya establecidas, probadas y disponibles actualmente. Si bien los resultados de la tesis pueden ser usados independientemente de OOHDM, introducimos el método como marco para razonar acerca de aplicaciones aprovechando la explícita separación en capas que ofrece, como veremos más adelante.

Seguimos analizando XML y las diversas especificaciones que conforman el concepto general de XML. Incluimos muchos ejemplos en esta sección y para no contaminar la esencia del trabajo, incluimos otros en el Apéndice donde se encuentra la implementación de los ejemplos.

Continuamos desarrollando el concepto de Vistas como forma de personalizar la información de acuerdo a la necesidad de la aplicación que la recibe o administra, perfil del usuario o simplemente mostrar la información de la manera que se espera.

Finalmente presentamos cuatro patrones como soluciones específicas usando XML.

Capítulo 2

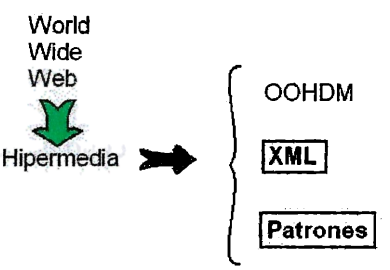
Hypermedia, OOHDM y patrones

Introducción

En la actualidad se plantea la necesidad de generar aplicaciones hypermedia en donde el usuario navega a través de sitios Web tomando un rol protagónico mientras que realiza transacciones complejas en la base de información. Esto implica la actual necesidad de interacción entre aplicaciones que hasta ahora no tenían que interactuar, es más, una desconocía totalmente a la otra. En el diseño no se consideraba la posibilidad de que la aplicación pudiera comunicarse con otras aplicaciones desconocidas al momento del diseño. Y si se pensaba, la solución era muy rudimentaria como por ejemplo llevar todos los datos a formato plano de texto.

Hasta ahora no se diseñaban teniendo en cuenta de dejar una puerta abierta para que en el futuro otra aplicación desconocida pudiera intercambiar datos con ellas.
Hoy esta capacidad de comunicación entre aplicaciones es una obligación.

Además con la evolución de ~~www~~ web, las aplicaciones hoy en día son constantemente modificadas (ver el trabajo "An Object Oriented Approach to Web-Based Application Design" [Rossi+98]). Por ende se necesita usar un método orientado a objetos para llevar adelante con éxito el diseño de aplicaciones basadas en la Web.



No es fácil desarrollar un diseño de una aplicación hypermedia así exitoso.
Las herramientas que ayudan son OOHDM (para lograr un diseño exitoso), XML para no transformarse en una isla, sino por el contrario compartir información y los patrones para usar el diseño probado de la solución de un problema y aprovechar la experiencia de otros que ya se enfrentaron con el problema.

Hipertexto e hipermedia

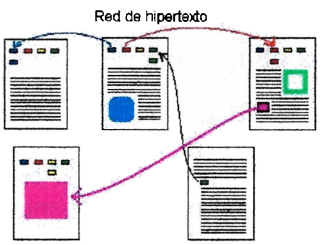
¿Qué es hipertexto?

Hipertexto es datos que contienen links a otros datos.

Un ejemplo cotidiano de hipertexto es una enciclopedia. Si se busca "radiación infrarroja" en una enciclopedia, al final de la explicación aparece una indicación de que se puede consultar el término radiación para obtener más información.
La World Wide Web (también llamada www o simplemente Web) actúa de la misma manera.

Encarta 1997 de Microsoft, define hipertexto en informática como "una metáfora para representar información en la cual texto, imágenes, sonidos y acciones se relacionan mediante links en un telaraña compleja, no secuencial de asociaciones que permiten al usuario hojear tópicos relacionados sin importar el orden de presentación de los tópicos".

Hipertexto es la organización de información de forma tal que no induce al lector a leer en orden secuencial ni de izquierda a derecha (como los textos tradicionales). Con hipertexto no se impone al lector un único orden de lectura. El autor de la estructura en forma de hipertexto ofrece muchas alternativas para completar la lectura.



Hipertexto tiene básicamente nodos y links. Un link forma una conexión entre nodos. Se forma de esta manera una red de nodos y links. Los links se asocian a anchors, que son regiones de un nodo. El usuario usa programas especiales denominados browsers para navegar por aplicaciones hipertexto.

El autor debe diseñar la estructura hipertexto de manera tal que el lector tenga la mayor información posible a su alcance, para que así pueda decidir si explora o no la información. Es necesario obtener el máximo provecho de la estructura de estos documentos, de modo que el usuario se obtenga beneficios y placer navegando la red de hipertexto. El diseño debe permitir navegar la red de hipertexto fácilmente y por sobre todo, debe facilitar que el usuario encuentre lo que busca rápidamente y sin perderse en la estructura.

Cuando el texto no es el único medio para ofrecer información, sino que además ésta se ofrece mediante imágenes, sonido y video estamos en presencia de multimedia.

El hipertexto que contiene multimedia, se llama hipermedia.

Aunque hipertexto implica que los elementos básicos son texto, los términos hipertexto e hipermedia a menudo se usan como sinónimos. Se amplía el concepto de hipertexto de manera tal que el de hipermedia se usa para referir a hipertexto donde además de información en forma de texto aparece información en otros medios o formatos.

Hipermedia y el desarrollo de aplicaciones en la Web

Hipermedia y la Web están estrechamente relacionadas. El advenimiento de la www (World Wide Web o Web simplemente), introdujo una generación de aplicaciones con características propias que deben estar preparadas para adaptarse al paradigma hipemedia. En [Rossi+98] se sugiere que "Las buenas aplicaciones basadas en la web deberían ser, ante todo, buenas aplicaciones hipermedia.", mientras que ya por 1997 estaba la idea de que "La World-Wide Web está basada en Hipermedia" [Arvanitis 97].

En consecuencia, entendiendo la estructura de navegación en hipertexto, y por extensión en hipermedia, se puede identificar las características de diseño apropiadas para aplicaciones Web.

Debido a su estructura de hipertexto, su complejidad aumenta, y por ende su diseño es más complejo y requiere de más esfuerzo, conocimiento y herramientas para que el resultado sea aceptable.

Históricamente se produjo un giro en el paradigma de la computación: antes se construía computadoras con el único propósito de procesar datos pero ahora las computadora no sólo se usan como máquinas de calcular, sino que estamos en una cultura en donde la computación es un medio de comunicación y simulación. Las computadoras sirven hoy tanto para acceder y comunicar conocimiento como también para construir "intrincados modelos de aspectos complejos de nuestro mundo." [Arvanitis 97], refiriéndose a P. J. Denning and R. M. Metcalfe (eds.), Beyond Calculation: The Next Fifty Years of Computing, Springer-Verlag, 1997 y a M. Weiser, The computer for the 21st century, Scientific American, 1991; 94-104 respectivamente.

Mientras tanto la World Wide Web crece día a día y cambió los hábitos de los usuarios de computadoras. Cada día se incorporan a la Web nuevas aplicaciones y estas aplicaciones basadas en la Web son aplicaciones hipermedia.

El importante crecimiento (podríamos decir boom) de la web y posteriormente el e-commerce lleva a que muchas aplicaciones se conciban con el propósito específico de ejecutarse sobre la red. Un ejemplo de esto es WebDB de Oracle una aplicación desarrollada para permitir crear y administrar sitios Web cuyos objetos se almacenan en la Base de Datos Oracle.

Por otro lado el usuario que navega tiene un rol protagónico tal que obliga a adaptar el diseño en función de él. Por ejemplo el diseño debe ser amigable, debe permitirle encontrar rápidamente lo que busca. El diseño debe ser simple [Nielsen99],

coherente y evitar cognitive overhead (que el usuario se maree y pierda la noción de dónde está) [Rossi+98].

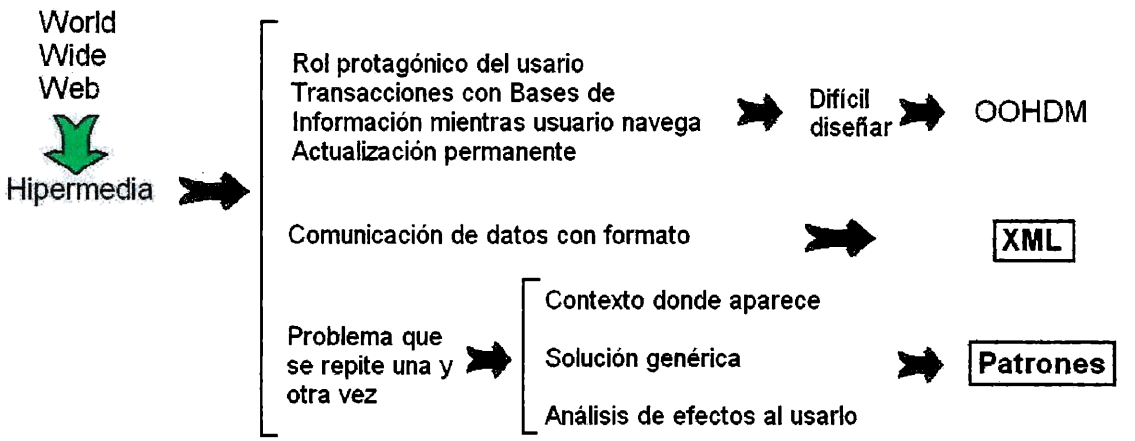
Por lo tanto el diseño debe tener en cuenta las necesidades del usuario por ejemplo compartir datos entre aplicaciones o que varias aplicaciones trabajen en colaboración para realizar una transacción para él.

La mayoría de los usuarios ya incorporaron en su uso cotidiano de la computadora la metodología de trabajo en red. Es sumamente común tener algo compartido en la red, usar un recurso compartido (por ejemplo la impresora), realizar las copias de resguardo en el disco del servidor local, usar la base de datos que se encuentra en el servidor, usar otro servidor para recibir o enviar mail o hacer una copia de resguardo en un servidor de la Web.

Cada vez aparecen más aplicaciones diseñadas para correr sobre la Web, ya sea una intranet (por ejemplo para que los empleados se enteren de las novedades internas de la empresa) o usando internet.

Por eso para tener aplicaciones hipermedia basadas en la Web hay que ajustar el diseño a los requerimientos modernos de la web. Esto implica considerar que internet tiene estructura cliente servidor, que el diseño debe estar preparado para que la aplicación se actualice constantemente, que la performance de la aplicación debe ser alta, el diseño amigable (en general es rentable que el usuario vuelva a visitar la aplicación), que la puesta en funcionamiento y actualizaciones deben llevarse a cabo en breves lapsos y manteniendo absoluta consistencia de los datos.

Hipermedia, mediante el desarrollo de la Web demostró ser indudablemente útil. Pero también se hizo evidente que para diseñar aplicaciones Web que soporten los requisitos de ésta es necesario utilizar herramientas para el diseño como OOHDM y utilizar una nueva manera de intercambiar información: XML



- Relación entre Hypermedia y XML

Cuando se piensa en diseñar aplicaciones basadas en la Web, la misma naturaleza de la Web sugiere usar objetos e hipermedia. Pero además la tendencia es que las aplicaciones sean capaces de comunicarse entre sí por más distintas que sean y aunque corran sobre plataformas distintas también. Teniendo eso en cuenta y para tener un medio para compartir datos se desarrolló XML.

Una de las ventajas de hipertexto y por extensión de hipermedia es que permite darle una **estructura jerárquica a la información**. Hipermedia permite la organización de múltiples jerarquías de información. En XML los datos se estructuran jerárquicamente de manera natural. Aunque XML estructura la información en forma de árbol, es flexible en ese aspecto, dando libertad al diseñador al momento de diseñar la estructura de la información. Por ejemplo el diseñador tiene la capacidad de decidir si habrá mucha anidación o no.

Ej a), información muy anidada:

```
<datos>
  <dato>
    <registro>
      <campo>1234
      </campo>
    </registro>
  </dato>
...
</datos>
```

Ej b) representación de la información con poca anidación

```
<informacion>
<dato/>
<dato/>
...
</informacion>
```

Hipertexto se caracteriza por la facilidad de trazar referencias, mientras que la familia de especificaciones XML tiene un miembro, XLink pensado especialmente para facilitar y mejorar del diseño en ese aspecto.

Hipertexto permite modularizar la información, (en consecuencia disminuye la necesidad de tener material duplicado), conformando una red de nodos conectados, mientras que los datos XML se pueden ver como información que se traslada de un nodo a otro. Desde este punto de vista un documento XML es data viajando a través de una red de agentes de procesamiento. Entonces tanto hipertexto como XML permiten modularizar la información.

Fundamentos de OOHDM

Si bien el resultado de la tesis pueden ser usados independientemente de OOHDM, presentamos una introducción al método como marco para razonar a cerca de

aplicaciones en la Web. OOHDM aporta la visión de reconocer capas abstractas separadas en el desarrollo de aplicaciones hipermedia. Las capas a las que nos referimos son los diseños Conceptual, de Navegación, de Interfase Abstracta y por último la Implementación.

Introducción

El diseño de aplicaciones hipermedia en la Web es complejo debido a las características específicas de la conjunción de www, hipermedia y aplicaciones. Por ejemplo la navegación es una característica distintivas de las aplicaciones basadas en la Web y hay que considerarla especialmente al momento de realizar el diseño.

Otras características que no podemos obviar son:

- **Aplicaciones constantemente modificadas** debido al crecimiento constante y evolución natural de la Web que obligan a incorporar nuevos servicios a las aplicaciones permanentemente.
- La obtención de información es realizada por el **usuario navegando** a través distintos sistemas de información. A medida que navega, el usuario realiza consultas o modifica la información y el espacio en donde se encuentra. Por ejemplo el usuario puede consultar la disponibilidad de pasajes en una línea de barcos y también realizar reservas. También el usuario puede estar en una aplicación en la Web y siguiendo un vínculo con un simple click puede saltar hasta el corazón de otra sin pasar por la página principal de esta nueva aplicación visitada [Nielsen99] lo que obliga a considerar en el diseño de que el usuario no entra al sitio sólo por la página principal, sino por cualquier parte.
- El **usuario tiene el control** y se puede predecir muy poco lo que hará o del medio que utilizará para acceder la información. Por ejemplo, al momento de diseñar la aplicación, el diseñador no se sabe el tamaño de la pantalla que usará el usuario, por lo tanto debería definir los elementos mostrados en relación al tamaño de la pantalla que tiene el usuario (por ejemplo definiendo tamaños de tablas con porcentajes).

El ambiente de desarrollo de aplicaciones en la Web sufre una conjunción de circunstancias que afectan el desarrollo de software y llevan a tener código difícil de mantener, no reusable e hipermedia embebida en diseños pobres. En estos desarrollos (y en otros desarrollos de software también, pero aquí demuestran ser características determinantes). El tiempo siempre es poco, las presiones económicas son altas, la demanda por la creación de aplicaciones sobre la web es alta y los desarrolladores a menudo consideran su tarea como la "noble artesanía de crear interfases Web". [De Muynck00]

Adaptar la aplicación al paradigma hipermedia no es sólo diseñarla como si fuera a ejecutarse sobre un entorno no-Web y agregarle luego una capa de navegación. Por el contrario, **en el mismo diseño se necesita especificar la estructura de navegación, así como también el diseño de hipermedia siempre teniendo en cuenta el**

comportamiento de la aplicación. OOHDM es un método de diseño que tiene en cuenta estas dos dimensiones del diseño.

Definición

OOHDM, por Object-Oriented Hypermedia Design Method [Rossi+98] es un método orientado a objetos para facilitar el diseño de aplicaciones basadas en la Web.

OOHDM sugiere organizar el desarrollo de aplicaciones hipermedia como un proceso de cuatro actividades:

- Diseño Conceptual
- Diseño de Navegación
- Diseño de Interfase Abstracta
- Implementación

Así mismo, OOHDM se basa en cuatro pilares que definen el sentido de utilización del método. Estos pilares son:

1. La noción de que los objetos de navegación son vistas, en el sentido de las Bases de Datos, de los objetos conceptuales.
2. El uso de abstracciones apropiadas para organizar el espacio de navegación con la introducción de contextos de navegación.
3. La separación de aspectos de interfase de los aspectos de navegación.
4. La identificación explícita de que hay decisiones de diseño que necesitan ser hechas en el momento de la implementación.

Cómo se usa OOHDM

Este método se usa desarrollando las cuatro actividades previamente mencionadas. En cada fase se genera un modelo elaborándose de esta manera cuatro capas bien definidas, tres de las cuales forman parte del diseño y una es la implementación.

Diseño Conceptual

La fase de Diseño Conceptual es el punto de inicio y consiste en modelar el dominio de la aplicación. Se conforma así un universo de discurso.

El Modelo Conceptual que se crea tiene dos tipos de objetos. Uno de estos tipos está conformado por los nodos que van a pertenecer al modelo de Navegación. El otro tipo de objetos está compuesto por objetos que pertenecerán al soporte computacional de la aplicación.

En este modelo se separan los objetos de navegación de los objetos conceptuales. La diferencia es que los objetos de navegación son objetos "customizados", hechos a la medida del perfil y las tareas que realiza el usuario.

Los objetos de navegación, por otra parte, tienen atributos que pueden ser los atributos de varios objetos conceptuales diferentes. Además de tener su propio comportamiento con funcionalidad de navegación, pueden tener el comportamiento para realizar actualizaciones y otras tareas.

Diseño de Navegación

En este modelo se estructura el espacio de Navegación con objetos de navegación. Para esto se crean Contextos de Navegación. La tarea consiste en agrupar los objetos del espacio de Navegación en conjuntos denominados Contextos de Navegación.

Además se especifican las transformaciones que sufrirá en espacio de Navegación con lo cual se define cómo se llevará a cabo la navegación. Por ejemplo dado un estado del espacio de Navegación, se define cuáles serán los objetos a los que se podrá navegar desde allí. El usuario no percibe los objetos de Navegación, sino que los accede vía objetos de interfase.

Diseño de Interfase Abstracta

En este modelo se especifica los objetos que el usuario percibirá, qué objetos de interfase van a activar la navegación, cómo se van a sincronizar los objetos de interfase multimedia y las transformaciones de interfase que podrán ocurrir.

Implementación

En esta fase se asocian los objetos conceptuales, los objetos de navegación y los objetos de interfase a un entorno de ejecución específico. En consecuencia depende del ambiente de implementación y de la plataforma de implementación. Como este es el momento en donde el diseño se implementa, al llegar a este punto el diseñador ya tiene definidos objetos que conforman el dominio del problema. También tiene establecido cómo se organizan estos objetos en función del perfil del usuario y de las tareas que realiza. Ya sabe cómo será la apariencia de la interfase y tiene definido el comportamiento.

Ahora tiene que definir cómo almacenar los objetos conceptuales y de navegación. También tiene que implementar la apariencia de la interfase con HTML y quizás tenga que usar programación para completar la funcionalidad. Por ejemplo incorporar código en VBscript para desarrollar alguna tarea.

Cuando se separaron los objetos de navegación de los objetos conceptuales, dijimos que la diferencia era que los objetos de navegación eran objetos "customizados",

hechos a la medida del perfil y las tareas que realiza el usuario. Sin embargo, OOHDM no especifica cómo implementar la tarea de tener en cuenta los perfiles del usuario y las tareas que desarrolló. Esto es porque depende del escenario en donde se esté utilizando OOHDM.

Para implementar los objetos que el usuario percibirá, es decir los del modelo obtenido del Diseño de Interfase Abstracta, se los puede por ejemplo llevar a campos de una tabla de una Base de Datos. En este caso los métodos asociados con las clases podrían ser implementados con procedimientos en la Base. Y si la base lo permite (por ejemplo teniendo Java incorporado a su motor), podrían mantenerse los objetos como tales con sus métodos y clases, mientras que la Base se encargaría de su almacenamiento.

Para implementar los contextos se requiere almacenar información de los estados. Por ejemplo en un momento dado, hay que reconocer cuál es el ítem "siguiente". También ha y que almacenar el comportamiento deseado ante una acción del usuario. La técnica a usar dependerá de la plataforma y del ambiente. Por ejemplo para mostrar los elementos se usa HTML, para almacenar información acerca del estado se pueden usar cookies, mientras que para la programación del comportamiento se puede usar jscript, Vbscript, Java,o el lenguaje que mejor se adecue al sistema.

Es justamente esta parte del proceso una de las más influenciadas por el desarrollo de la tecnología (otra también influenciada es el Diseño Conceptual). Es en estas actividades del proceso donde XML muestra sus beneficios.

Ventajas en el uso de OOHDM

Entre otras ventajas que aparecen en [Rossi+98] se encuentra que "El uso de OOHDM y de patrones de diseño redujo el tiempo de desarrollo y permitió discutir diseños antes de tener realmente implementaciones ejecutándose. Otro beneficio de los modelos OOHDM fue mejorar la comunicación con clientes y con profesionales no relacionados con la computación como diseñadores gráficos y especialistas en marketing que formaban parte del equipo de diseño".

Además "separar los intereses entre objetos conceptuales, de navegación y de interfase ayuda a generar aplicaciones que son fáciles de extender y de mantener.

Patrones

En esta sección definimos el concepto de patrón, planteamos los beneficios que aporta usar patrones y analizamos el formato que en general toman. Finalmente observamos los patrones desde el punto de vista de XML, la visión que afectará el resultado de este trabajo.

Qué es un patrón?

> Se puede ver un patrón como un camino probado y que conduce a buenos resultados para tratar un problema que aparece una y otra vez. [XMLPAT]

Un patrón es, según su creador Christopher Alexander "una regla de tres partes la cual muestra la relación entre un contexto dado, un problema y una solución".

Refiriéndose a patrones en la construcción y ciudades, pero también válido para el diseño de patrones Orientado a Objetos dice que "Cada patrón define un problema que ocurre una y otra vez en nuestro ambiente, y luego describe la esencia de la solución a ese problema, de tal forma que puedes usar esa solución un millón de veces, sin hacerlo dos veces de la misma manera. [Gamma+95]

Los patrones sistemáticamente evalúan, nombran y catalogan diseños recurrentes. De esta manera, cuando los diseñadores necesitan crear soluciones a los problemas de diseño, los patrones les permiten basarse en diseños flexibles, elegantes y probados.

El diseño de patrones implica generar soluciones reusables, para que quien consulte esos patrones pueda fácilmente detectar qué patrón le conviene usar y obtenga de él el máximo beneficio, aprovechando la experiencia de alguien que ya transitó ese camino.

Beneficios del uso de patrones

> Los patrones de diseño resuelven problemas de diseño. Esta idea aparece en el clásico libro de Patrones de Diseño . [Gamma+95]
> Por lo tanto conocer patrones es como tener la alacena con soluciones.

El diseñar soluciones desde cero implica muchas veces transitar caminos que no llevan a una solución válida para el problema, obligando a intentar otro camino una vez que se detecta el error y con la pérdida de tiempo y esfuerzo que implica. Muchas veces

el conocimiento y desarrollo se basa en el conocimiento que nos pasan otros que ya se enfrentaron a nuestro problema.

Por lo tanto los patrones ahorran el tiempo y esfuerzo si los comparamos con diseñar soluciones sin ellos. Permiten aprovechar la experiencia de otros al leer soluciones generales para contextos determinados (y definidos en los mismos patrones) que ya inventaron otras personas. Podemos entonces construir soluciones basadas en esa experiencia.

Formato de patrón

Un patrón se trata de la solución a un problema en un contexto. Por eso tiene cuatro componentes esenciales:

- Nombre
  Para ubicarlo fácilmente. Nos permite expresarnos en un nivel de abstracción más alto y referirnos rápidamente a todo el conjunto de componentes del patrón. De esta forma nos referimos puntualmente a un patrón en documentos, creando soluciones, etc. Se pone un nombre a los conceptos adquiridos para que sea más fácil incorporarlos en una charla o en un intercambio de ideas.
- Problema
  Describe el problema y el contexto donde ocurre. Sirve para reconocer cuándo aplicar el patrón. Puede incluir una lista de condiciones que deben darse para que tenga sentido usar el patrón.
- Solución
  Descripción abstracta los elementos que conforman la resolución del problema y cómo están organizados. No corresponde a una implementación concreta, particular, específica; sino que es el núcleo de la solución al problema. Es el concepto expresado de forma tal que se pueda usar varias veces y con la flexibilidad de que cada vez se use de manera distinta, como se usa un template.
- Consecuencias
  Son los factores que se verán afectados con el uso del patrón. Los resultados de aplicar el patrón, con un análisis de las ganancias y las pérdidas que se esperan en caso de aplicarlo. Por ejemplo si afectará la flexibilidad o portabilidad del sistema, así como también asuntos de implementación.

Observación: el concepto de patrón es, en este capítulo de este trabajo, el de Design Patterns, Elements of Reusable Object-Oriented Software de Gamma, Helm, Johnson y Vlissides. Luego se adaptará a la tecnología y demás elementos modernos. Simpre con el objetivo de "resolver un problema de diseño general en un contexto específico"

Por sobre todo el patrón tiene que ser entendible y reusable Para esto hay que describirlo de manera precisa explicando el objetivo, contexto, factores que afectan el diseño, un análisis de beneficios y asuntos que hay que resignar, la solución propuesta, factores de implementación y observaciones que el diseñador considere útil para que se haga un mejor uso del patrón.

Por eso al consultar distintas fuentes, los ítems presentes en la descripción del patrón difieren. Por ejemplo en [Gamma+95] los autores usan un formato consistente para todos los patrones. Para esto crean un template con una estructura uniforme para todos. Es decir que los patrones tienen una estructura conformada por secciones. En este caso las secciones y su significado son:

Nombre	Comunica la esencia del patrón. Es vital porque formará parte del vocabulario de diseño de quienes lo usen.
Objetivo	Es la respuesta a ¿qué hace el patrón? o ¿a qué problema o asunto de diseño apunta?
Alias	Otros nombres por los cuales se conozca al patrón. Puede no tener alias.
Motivación	Un escenario que ilustra un problema de diseño y cómo las clases y las estructuras de objetos en el patrón resuelven el problema.
Aplicabilidad	¿En qué situaciones se puede aplicar el patrón de diseño? ¿Qué situaciones tienen soluciones ingenuas en problemas a donde apunta el patrón?
Estructura	Es una representación gráfica de las clases que aparecen en el patrón.
Participantes	Las clases y/o objetos que participan en el patrón de diseño y sus responsabilidades.
Colaboraciones	Cómo colaboran los participantes para cumplir con sus responsabilidades.
Consecuencias	Cómo trabaja el patrón para cumplir con sus objetivos. Cuales son los resultados de usarlo, cuáles son las ventajas y qué hay que tolerar para tenerlas.
Implementación	Técnicas o pistas que hay que conocer al momento de implementar el patrón. Se considera si el lenguaje afecta algo.
Ejemplo de código	Una parte del código para mostrar cómo se podría implementar el patrón.
Usos conocidos	Ejemplos del patrón encontrados en sistemas reales.
Patrones Relacionados	Si el patrón en cuestión se relaciona con otros, entonces se marcan las diferencias que sean importantes y se indica también si este patrón debiera ser usado junto a otros.

Por otra parte en Patterns for e-commerce applicatios [Rossi+00] los patrones también comparten la estructura común entre sí, pero las secciones de esta estructura son:

Nombre	muy significativo porque se incorpora al vocabulario como referencia al patrón.
Objetivo	muy concisamente se informa qué hace el patrón.
Motivación	contexto en el que cobra sentido y en el que se ubica el patrón. Puede

	tener ejemplos para ayudar a entender la descripción más abstracta que viene después en el patrón.
Fuerzas	puntos que el patrón debe cumplir o solucionar y condiciones que se imponen debido a la situación en que suele presentarse el problema que soluciona el patrón.
Solución	elementos y su interacción para lograr el objetivo. Es una descripción de la resolución del problema concisa y clara de una granularidad media que puede ir acompañada por diagramas.
Ejemplo	sitios reales en donde aparece el patrón junto a una breve descripción.
Consecuencias	ventajas y desventajas del uso del patrón.
Implementación	indicaciones de cómo implementar la solución. Puntos del contexto que hay que adaptar, complejidad esperada de la implementación o guías para la implementación.

Observación: esta es la estructura que seguirán los patrones y las soluciones presentados en este trabajo. En algunos casos, posiblemente debido a la novedad del tema XML, los patrones no tienen en la sección Ejemplo ejemplos reales. Para facilitar la comprensión se exponen ejemplos creados especialmente.

En http://www.xmlpatterns.com [XMLPAT] el formato de los patrones consiste en las secciones nombre, resumen, problema, contexto, fuerzas, solución, ejemplos, discusión, patrones relacionados y usos conocidos. En este sitio se observan patrones catalogados en un intento por " comenzar a generar un lenguaje de patrones estructurales de XML".

En general los patrones, aunque varíe un poco su formato e independientemente del nombre de sus secciones, tienen la descripción del problema y el contexto donde ocurre, la solución de diseño propuesta y ejemplos reales de su uso (si existen). Un ejemplo de esto son los patrones de navegación para mejorar sistemas de información en la Web como se observa en [Schwabe99].

Patrones XML

En la familia XML, como en muchas otras áreas, se pueden definir patrones con diferentes niveles y en varios aspectos.
Respecto al asunto de los niveles, se pueden definir patrones con mayor o menor grado de abstracción. Por ejemplo se podría buscar patrones que mostrarán cómo enfrentar el diseño de manera global. Estos patrones indicarían en qué casos conviene usar XML. Otro nivel de patrones estaría conformado por patrones que aportan soluciones a problemas más específicos y mostrarían más en detalle la manera de usar XML y, por supuesto, qué miembro de la familia XML usar.

En cuanto a los diversos aspectos según los cuales podemos definir patrones, se trata de patrones específicos que aportan soluciones a la hora de usar XML en el diseño. Por ejemplo considerando la familia XML podríamos desarrollar patrones para:

- darle formato a la data
- darle formato al documento
- utilizar stylesheets
- usar el DOM mediante lenguajes de programación

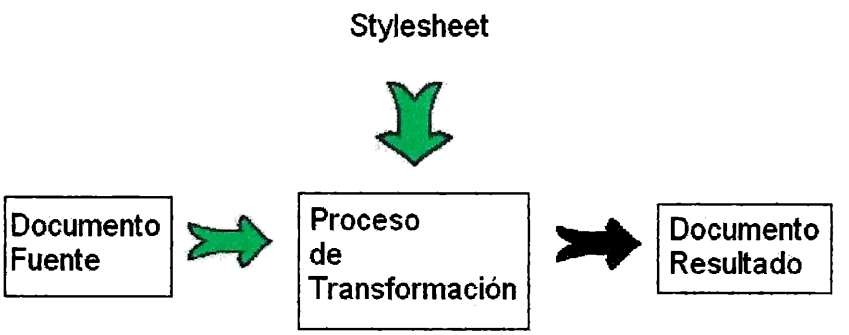
En cuanto a patrones XML para darle formato de datos y como formato de documentos, se basan en sugerencias de formato. Corresponde en este punto hacer una distinción entre el formato de los datos y los formatos del documento.

El formato de datos implica trabajar con XML manipulando objetos o datos. Por ejemplo al incorporar un subelemento al árbol XML se está incorporando un objeto, pero también se le puede estar incorporando un dato con una estructura determinada, es decir que estamos incorporando un elemento que puede tener otros elementos o atributos.

El formato del documento implica tomar la decisión de definir una estructura que indique cual es el formato "correcto" del documento. Por ejemplo crear un DTD.

Tanto en el caso de los patrones usados para darle formato a la data como al documento, se trata de sugerencias para diseñar el DTD o el formato de los datos (o el formato que llevará el árbol de datos XML). En este punto encontramos material. Esta área fue la primera en tener patrones. [XMLPAT] XML es útil para que los datos estén bien descriptos, pero un mal diseño en la organización de los datos en el archivo XML o en el DTD puede dificultar su almacenamiento, recuperación y transporte. Por ejemplo si la estructura de los datos es complicada y además no refleja la naturaleza, a la hora de incorporar y de recuperar los datos el programador tendrá que escribir mucho código complicado, esto hará difícil escribir y mantener el programa, al mismo tiempo que bajará la performance.

Los patrones que utilizan stylesheets responden a una estructura de procesamiento particular. En la siguiente figura observamos un Proceso de Transformación que recibiendo un Documento Fuente y una stylesheet produce un Documento Resultado.



El Documento Fuente tiene los datos respetando una estructura XML. La stylesheet tiene las instrucciones que indican al Proceso de Transformación cómo deben aparecer los datos en el Documento Resultado.

Los patrones que usan el DOM se basan en programar usando lenguajes de scripts o Java y el DOM (Document Object Model) para procesar datos de los archivos XML. Dependen en gran medida de la plataforma de implementación y por lo tanto en general son muy específicos de las circunstancias. (Decimos en general porque nada impide definir un patrón muy abstracto acerca del uso de XML con el DOM, aunque tendría más utilidad como explicación de para qué sirve XML que como patrón de diseño).

Contexto actual

Como mencionamos en la sección anterior, existen patrones para vocabularios [XMLPatterns]. Por supuesto que conviene conocerlos para incorporarlos al vocabulario, pero aunque es de esperar que día tras día se obtengan patrones nuevos en este aspecto, todavía falta mucho por descubrir.

Los patrones definidos en otros aspectos aparecen más tarde en la comunidad de patrones. Esto puede deberse a una cierta inseguridad acerca de la tecnología dado que las especificaciones de los miembros de la familia XML evolucionan constantemente.

No obstante XSLT se encuentra lo suficientemente establecido como para hacer patrones útiles y con perspectiva de seguir siendo útiles en el futuro. En este aspecto observaremos en el capítulo 5, donde se encuentran todos los patrones, un patrón que usa XSLT cuyo nombre es Group By (porque cumple la misma función que la sentencia homónima de SQL).

Independientemente de los aspectos en los que intentemos obtener patrones XML, en algún punto tenemos que incluir HTML para mostrar los elementos y muchas veces llamados a programas en el Web server.

Capítulo 3

La familia XML

En este capítulo exploramos XML desde el punto de vista práctico y didáctico. El propósito es obtener una idea acabada de qué es XML y cómo se puede sacar provecho de sus características. Veremos que XML es muy amplio, pero describiremos en detalle los temas que afectan el resultado de este trabajo, mientras que dedicaremos una mirada rápida a partes de XML que prometen ser útiles en el futuro. Incluímos muchos ejemplos con el objetivo de clarificar y hacer de este trabajo algo útil y práctico. Para no contaminar, en algunos casos hacemos referencia a ejemplos que aparecen en el Apéndice.

Primero tomamos contacto con las especificaciones más importantes del World Wide Web Consortium. Luego conoceremos el DOM, el modelo de objetos para XML que nos permite manejar la información que se encuentra en el árbol de cada documento XML. Consideramos que la extensión de este capítulo se justifica por la amplitud del tema XML, el enfoque práctico con que lo abordamos y, sobre todo, porque estos temas servirán para entender los patrones desarrollados en el capítulo 5.

Introducción

La familia XML es un conjunto de especificaciones que conforman un estándar que define las características de un mecanismo independiente de plataformas desarrollado para compartir datos.

Existe un organismo que emite especificaciones acerca de este mecanismo. Ese organismo es el World Wide Web Consortium y sus recomendaciones se encuentran en www.w3.org. Por lo tanto XML es un estándar internacional controlado por un organismo de la industria, no por una empresa particular.

Podemos considerar a XML como un formato de transferencia de datos multi-plataforma. Los participantes más importantes de la industria de la computación lo están adoptando, de hecho se lo usa cada vez más. Son esos mismos importantes integrantes de la industria de la computación los que crean herramientas para usar XML, invierten en XML y lo soportan. El hecho de que haya varias compañías importantes comprometidas con esta tecnología es bueno, porque significa que usar XML no implica estar atado a un único vendedor.

XML es una familia por que existen varias especificaciones que se pueden usar en equipo. Muchas veces se asocia al nombre XML elementos pertenecientes a otras especificaciones que están relacionadas con XML. Hay una especificación, la XML 1.0 [W3CXML], que define qué son los tags, elementos y atributos entre otras cosas. Analizaremos esta especificación en la sección siguiente. Alrededor de la especificación XML 1.0 existen otras que complementan la guía de uso de XML explicando cómo realizar otras tareas que no explica la especificación XML 1.0. Por ejemplo existe una recomendación XLink dedicada a describir una manera estándar de incorporar "hyperlinks" (links de hipertexto) a un archivo XML. XPointer es otra recomendación dedicada a señalar partes de un documento XML.

Hay un conjunto de llamadas a funciones denominado DOM, por Document Object Model. Como se verá más adelante, el objetivo del DOM es tener un estándar para manipular XML desde un lenguaje de programación.

También existe otra especificación llamada XML Namespaces para no confundir los nombres de los tags definidos en un documento XML con los de otro.

Finalmente analizaremos las recomendaciones XSL y XSLT, también pertenecientes a la familia XML y de gran utilidad para crear presentaciones de datos XML (stylesheets) y un lenguaje de transformación para reacomodar la estructura de los datos XML, por ejemplo incorporar atributos, eliminar etiquetas o mostrar datos de un documento XML en orden distinto al que están almacenados, etc.

Las recomendaciones son muy detalladas y evolucionan existiendo varias versiones de algunas. Todas se encuentran disponible en www.w3.org/TR, la página de reportes técnicos del cuerpo que organiza y lanza las especificaciones. No obstante en secciones posteriores veremos características importantes de las especificaciones más usadas.

La integración de aplicaciones basada en XML es una tecnología clave en transacciones business-to-business en la Web [TECHWEB].
XML no sea quizás siempre la mejor solución, pero vale la pena tenerlo en cuenta a la hora de planear un diseño. Veamos por qué es tan codiciado.

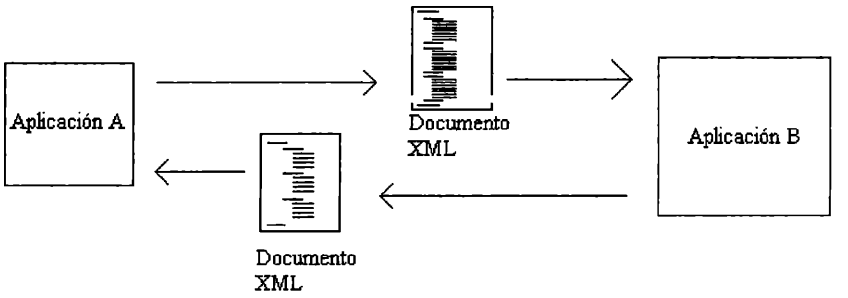
XML

XML es el acrónimo de eXtensible Markup Languaje, en esencia es un mecanismo de almacenamiento de datos y una herramienta para definir lenguajes. Tiene características especiales, por ejemplo XML tiene la habilidad de cruzar todos los límites entre plataformas. Además, como veremos en seguida, es un método para tener datos estructurados en un archivo de texto. Almacenar los datos como texto permite leerlos sin necesitar un programa especial que los recupere y produzca para que tengan un formato legible.

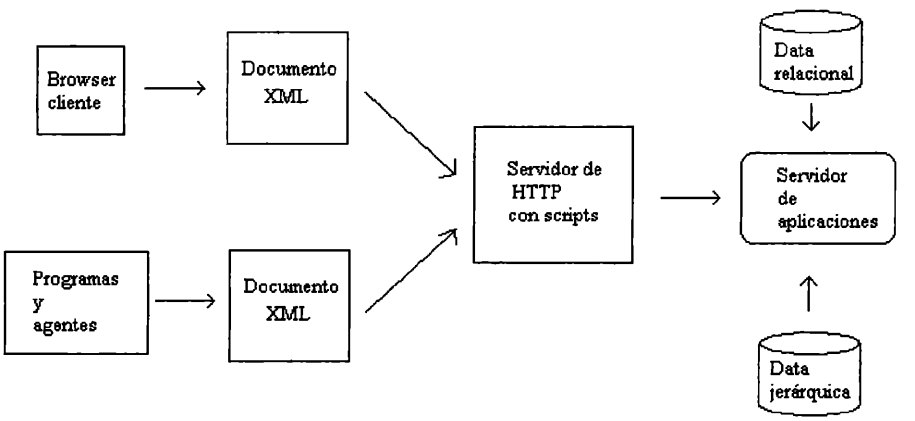
El conjunto de reglas o convenciones que impone la especificación XML, permite diseñar formatos de texto para los datos estructurados, haciendo que se almacenen de manera no ambigua, independiente de la plataforma y que en el momento de la recuperación se pueda verificar si la estructura es la correcta.

Las características principales de XML son:

- Es un mecanismo para *intercambiar datos.*



La particularidad de este mecanismo es que permite intercambiar datos entre programas que no tengan que coordinarse previamente y además en el mismo almacenamiento de datos en forma de texto se guarda la estructura de los datos, que también se intercambia junto con los datos. Por ejemplo supongamos que nos encontramos con una red cuyos nodos pueden ser clientes, servidores y aplicaciones. Supongamos que los datos viajan por esa red y que tienen un formato estándar. Usando el mecanismo propuesto por XML, documentamos la estructura esperada por cada nodo. Obviamente la estructura esperada por el nodo que recibe los datos debe ser la misma estructura que envía el nodo emisor. Con estas condiciones, el nodo que recibe los datos puede verificar que la estructura que recibió es la correcta apenas haberla recibido y XML permite documentar la estructura esperada por el servidor.
Es más, el cliente puede especificar formalmente la estructura del pedido usando un mecanismo estándar obtenido del servidor en *tiempo de ejecución.*

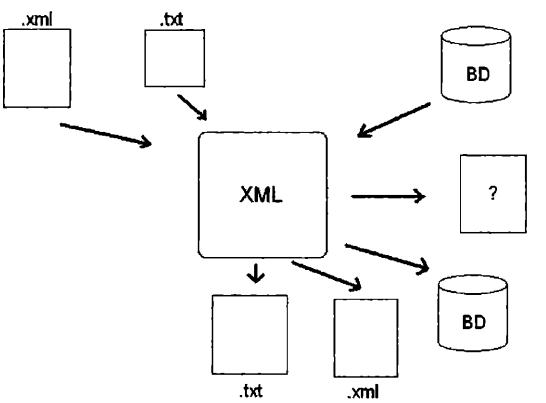


El acoplamiento entre la aplicación servidora y la cliente es considerablemente menor debido a la habilidad de un programa de descubrir la estructura de un documento XML.

XML permite serializar objetos, definiendo serializar como "el proceso de codificar un grafo de objetos interrelacionados cualquiera (como objetos Java) en secuencia lineal". La serialización es importante cuando hay que almacenar el estado de los objetos en un archivo, o enviarlos por una red. XML permite serializar los objetos llevándolos a una cadena de caracteres.

De esta manera la serialización se lleva a cabo sin depender de una sintaxis propietaria, porque (igual que Java) usa *Unicode*. Además en todo momento la información queda en un formato que una persona puede leer fácilmente. El formato XML en el que quedan los datos que fueron serializados se puede leer con cualquier editor de texto y con muy poco esfuerzo, porque se necesita aún para presentar los datos con un formato "customizado" para el usuario, sólo se necesita un programa o una stylesheet.
XML se puede mostrar en un browser y también se puede usar para pasar objetos de una aplicación a otra a través de una red. Por eso se está usando para compartir datos entre aplicaciones multi-tier. De esta forma el cliente recibe o muestra los datos a la medida del usuario, mientras que en la middle-tier se usa XML para intercambiar datos o serializar objetos.

- *Separa los datos de la representación* de los datos.
  XML usa un formato de almacenamiento de datos que los presenta listos para ser manipulados por programas. La presentación de esos datos no se limita a un browser (a diferencia de cuando un programa arma una página html, que es confeccionada con el fin de ser presentada por un browser, o una página ASP, que termina siendo una página html con los resultados del procesamiento embebidos en ella. En XML los mismos datos pueden ser preparados por un programa para ser presentados por un browser, preparados para entregárselos a un agente, almacenados en otro archivo XML o preparados para un dispositivo menos usual.
  En la figura vemos un sistema que por usar XML puede recibir datos desde varias fuentes , transformarlos a XML, procesarlos y entregarlos como salida hacia distintas fuentes, todo vía XML.
  El mecanismo de intercambio son los documentos XML gracias a que son secuencias de caracteres y no se involucra en los documentos nada que indique el destino final de los datos.
  Por ejemplo no se almacena información especial si serán vistos en un browser como Netscape, en un teléfono o si se almacenarán en una Base de Datos.



  Si el cliente requiere HTML, entonces se aplica una transformación de datos al documento XML para crear la página HTML, pero en todo caso la aplicación subyacente que creó los datos en XML no se modifica y el documento XML tampoco.

- Es un estándar con soporte para *todas las plataformas*.
  Además el software de soporte necesario es gratuito.
- Es *fácil de leer*
  Un documento XML es una combinación de datos y de significado de los datos

(markup). El significado se almacena en etiquetas. Las etiquetas que rodean al dato le dan su significado mientras que el dato caracter es el contenido.
Ejemplo a:
para indicar "este es el nombre del cliente", escribimos

```
<nombre_cliente>Juan Perez</nombre_cliente>
```

Ejemplo b:
supongamos que tenemos una Base de Datos de bibliografía y una tabla que almacena referencias a artículos y libros.

```
Libro
Identificador: Nielsen99
   Autor
         Apellido: Nielsen
         Inicial: J
Título: Designing Web Usability: The Practice of Simplicity
Editorial: New Riders Publishing
Fecha:1999
```

En XML podría crear un conjunto de tags para representar la información así:

```
<referencia>
   <!--libro-->
   <identificador>Nielsen99</identificador>
         <autor>
                     <apellido>Nielsen</apellido>
                     <inicial>J</inicial>
         </autor>
   <titulo>Designing Web Usability: The Practice of Simplicity</titulo>
   <editorial>New Riders Publishing</editorial>
   <fecha>1999</fecha>
</referencia>
```

La bibliografía de este trabajo de Grado se creó en un archivo XML con esta estructura de datos.

Como XML es un *formato de texto*, usa tags (etiquetas) para delimitar los datos. Esto hace que los datos reales aparezcan delimitados por nombres que tienen sentido para el lector humano. Se crean de esta manera vocabularios para cada área donde se use. Aunque muy bien una etiqueta podría ser `<qwertyksñgjjgh>` (y mientras que se respeten las reglas de sintaxis de la especificación de XML, el archivo XML está bien formado y el procesamiento continúa normalmente), no se usaría porque sería confuso para las personas. Es lo mismo que ocurre en la Bases de Datos en donde al motor de BD no le afecta si el usuario declara el nombre de la columna que almacenará las ciudades como "abracadabra" en vez de "ciudad", pero un mal nombre complicaría el desarrollo de la aplicación.

Ejemplo:
Nombre de etiqueta mal elegido. El dato es una ciudad

```
<nombre>La Plata</nombre>
```

Nombre bien elegido porque facilita la tarea al programador o diseñador

```
<ciudad>La Plata</ciudad>
```
Aquí el nombre de la etiqueta recuerda el significado del dato.

Sin embargo, XML está menos desarrollado con el objetivo de que sea leído por personas que HTML.

- Los documentos *describen su propia sintaxis*
Cada documento contiene su propio árbol parseado en el formato con el que se ubican los tags de sus elementos.
Todo documento bien formado (y como veremos más adelante bien formado en el contexto XML tiene un significado especial) tiene la forma de un árbol.
Su sintaxis se observa mediante dos maneras: una es por el formato del árbol que deben formar las etiquetas, otra es por la presencia de (o referencia a) la definición explícita del formato del documento.

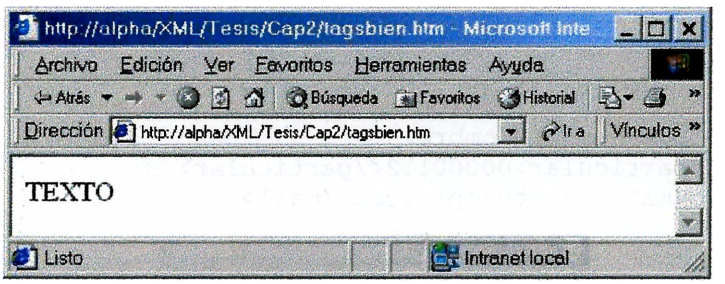
Los elementos de un documento XML deben tener etiquetas de finalización o estar escritos de una manera especial (si son elementos vacíos), y todos los elementos deben estar anidados. Es decir que un documento XML tiene un elemento que es la raíz. Todos los demás son sus hijos o descendientes más lejanos. Así mismo cada elemento tiene sólo un padre (excepto la raíz, que no tiene ninguno), conformando de esta manera un árbol.
Dicho de otra forma, todos los elementos tienen un tag de inicio y uno de finalización y si en orden Last In First Out apareamos los tags, cada par debe estar formado por un tag inicio con su correspondiente tag de finalización. Un programa que verifique esto es fácil de escribir usando una pila. Los programas que desarrollan esa tarea para documentos XML se llaman **parsers** y se usan para verificar que el documento está bien formado.

La sintaxis en el formato del documento en forma de árbol hace que los documentos XML sean fáciles de parsear. XML fue diseñado así para que el "usuario-agente" que interpreta el archivo pueda estar en una plataforma reducida como por ejemplo una PDA (Personal Digital Assistant).
Los documentos XML son archivos de texto para que sean ubicuos y para que los programadores puedan encontrar mejor los errores y puedan editar un documento XML para corregirlo usando simplemente un editor de texto. Porque la falta de una etiqueta esperada, por ejemplo, hace inutilizable el documento XML. Esto es debido a que en el diseño de XML se estableció que *no se tolerarían ambigüedades*. A diferencia de HTML, en donde se tolera que ante una etiqueta de apertura falte la correspondiente de cierre (por ejemplo la etiqueta de párrafo sin una simétrica), un programa que interprete un archivo XML no puede intentar adivinar qué quiso escribir el creador del documento XML, sino que el documento debe cumplir todas las reglas o considerarlo roto y descartarlo.

Ejemplo a: para dos tags a y b no es legal tener
```
<a>  ...  <b>  ...  </a>  ...  </b>
```
Sí es legal
```
<a>...
```

```
        <b>
              ...
        </b>
        ...
</a>
```

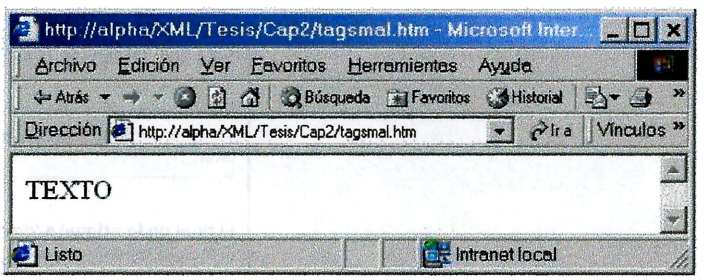
Ejemplo b:
usando HTML, un servidor de Web es capaz de mostrar este archivo (tagsBien.htm)




```
<HTML>
<BODY>
<P>TEXTO</P>
</BODY>
</HTML>
```

Pero también muestra este ,(tagsMal.htm) aunque falte la etiqueta P de cierre,

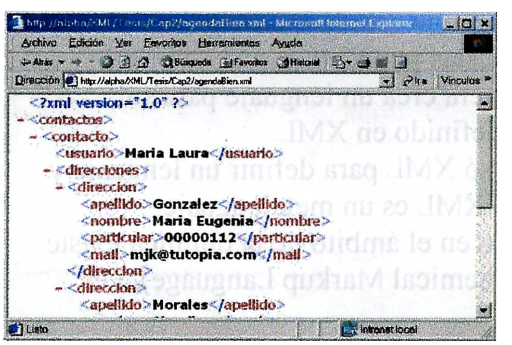
```
</P>
<HTML>
<BODY>
<P>TEXTO
</BODY>
</HTML>
```



Ejemplo c:
veamos cómo mostrar un documento XML bien formado y el error que aparece al intentar mostrar un documento XML que no está bien formado.

Bien formado:
El browser muestra este archivo bien (agendaBien.xml)

```
<?xml version="1.0"?>
<contactos>
     <contacto>
        <usuario>Maria Laura</usuario>
        <direcciones>
        <direccion>
           <apellido>Gonzalez</apellido>
           <nombre>Maria Eugenia</nombre>
           <particular>00000112</particular>
           <mail>mjk@tutopia.com</mail>
        </direccion>
         ...
        </direcciones>
     </contacto>
</contactos>
```



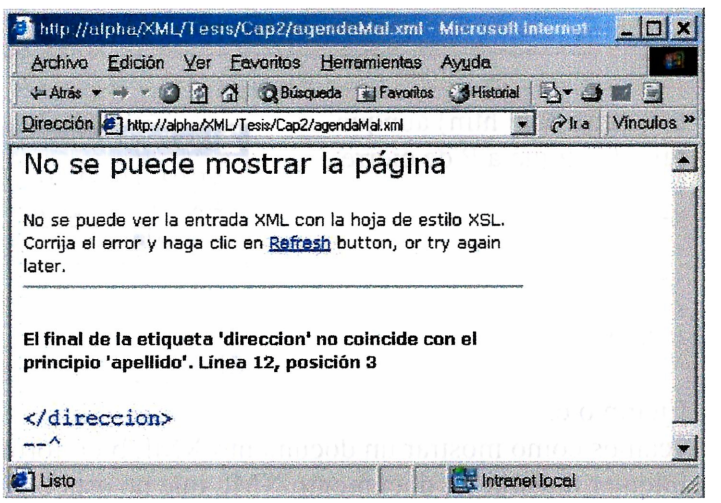
El código completo del archivo agendaBien.xml se encuentra en el Apéndice, Implementación de los

ejemplos, Ejemplo de documento XML.

Documento que no está bien formado:

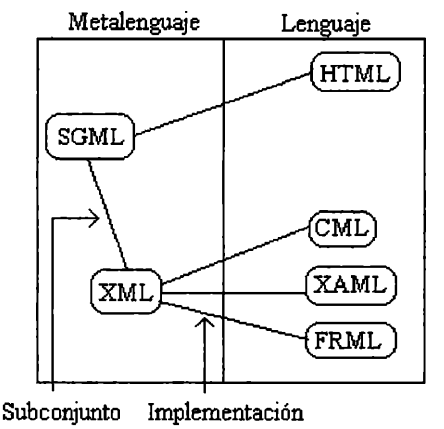
A este archivo (agendaMal.xml) le falta la etiqueta </apellido> de cierre.

```
<?xml version="1.0"?>
<contactos>
   <contacto>
         <usuario>Maria Laura</usuario>
         <direcciones>
         <direccion>
            <apellido>Gonzalez  <!--falta la etiqueta de cierre -->
            <nombre>Maria Eugenia</nombre>
                     <particular>00000112</particular>
                      <mail>mjk@tutopia.com</mail>
         </direccion>
         ...
         </direcciones>
   </contacto>
</contactos>
```



Por eso el browser IE5 no muestra los datos e indica un error.

Es una *herramienta para definir lenguajes Markup.*
Como observamos en los ejemplos, las etiquetas deben cumplir reglas estrictas, pero no están predefinidas. A diferencia de HTML en donde <B> tiene un significado, en XML las etiquetas que encierran a los datos no significan nada para el browser, sí para las personas. Esto permite que el diseñador elija nombres memotécnicos y que se creen vocabularios específicos.
Por lo tanto, el diseñador define conjuntos de etiquetas y conjuntos de atributos para cada etiqueta.
Pero además define *reglas para acomodar las etiquetas en el documento*, esto dará la estructura de los datos finalmente. Por supuesto que todo esto lo hace siguiendo reglas impuestas por XML.
De esta manera crea un lenguaje para su aplicación definido en XML.
O sea que usó XML para definir un lenguaje y por lo tanto XML es un metalenguaje.
Por ejemplo en el ámbito de la química existe un CML (Chemical Markup Language), ver



http://www.xml-cml.org desarrollado con XML. También hay un lenguaje de matemáticas llamado MathML (Mathematical Markup Language (MathML) Version 2.0) que brinda una manera estándar de escribir e intercambiar expresiones matemáticas. Para más información ver http://www.w3.org/Math/ . Mientras tanto la NASA tiene IML (Instrument Markup Language) para el control de instrumentos de laboratorio. AIML (Astronomical Instrument Markup), que se encuentra en http://pioneer.gsfc.nasa.gov/public/aiml/.

XML se transformó en una tecnología para la estandarización de formatos de datos y de documentos.
Día tras día se anuncia el desarrollo de especificaciones basadas en XML. Un ejemplo de esto es la "especificación basada en XML para la preparación y el intercambio de reportes financieros y datos". Esta especificación se llama XFRML, por "XML-based Financial Reporting Markup Language", el que será el lenguaje digital para los negocios.
Otro ejemplo es XAML, Transaction Authority Markup Language una especificación que están desarrollado en conjunto Bowstreet Inc., Hewlett-Packard Co., IBM Corp., Oracle Corp., and Sun Microsystems Inc.para asegurar la integridad de transacciones automáticas en Internet.

XML provee los medios para formular estándares fácilmente, porque brinda un marco donde las reglas son comunes para todos, fáciles de entender y no son dependientes de la plataforma por lo tanto es fácil discutir y formular un estándar. La estandarización de formatos de datos provee grandes beneficios, por ejemplo la facilidad para intercambiar información.
Es de esperar que paulatinamente se vayan poblando todas las áreas en donde se intercambie información con lenguajes XML (computación, medicina, etc.).

Etiquetas y significado

También podemos ver XML como un conjunto de reglas que definen cómo asignar significado a textos o documentos. "Marking up" un documento es el proceso de identificar ciertas áreas del documento y asignarles significado. En algunos casos ese significado que se asocia a ciertas áreas es para darles formato y en otros no.

Por ejemplo en HTML, la etiqueta (o tag, en inglés) `<B>` indica al browser que el texto que sigue debe estar en negrita. En este caso tenemos un tag que da formato al texto. Por otra parte el tag `<BODY>` informa que a continuación viene el cuerpo del archivo, pero no establece un formato.

XML y HTML usan tags, pero XML desde una perspectiva diferente. Porque XML está diseñado para describir la estructura del texto, mientras que HTML se orienta a cómo se debería mostrar.

A diferencia de HTML, XML no tiene un conjunto de tags fijo, por el contrario, los tags se crean explícitamente para el documento o la aplicación y en función del significado de los datos. Como los tags en XML no implican formato y no pertenecen a

un conjunto fijo, para el programa que muestra el documento XML no significan nada. Entonces el programa que los muestra (por ejemplo el browser) no tiene predefinido hacer otra cosa que mostrar los tags como los encuentra.

Por ejemplo si encuentra el tag `<H1>`, lo muestra de la misma manera que mostraría `<titulo>`, no le va dar formato de encabezado como haría si encontrase el tag `<H1>` en un archivo HTML.

En conclusión los tags pueden llevar el nombre que el diseñador quiera (respetando unas pocas reglas que veremos en la siguiente sección). Es lógico usar nombres con sentido, para que describan los datos, sobre todo considerando que un archivo xml es texto y se puede leer como tal.

Por ser texto estándar, es fácil transferir XML entre aplicaciones y como la ubicación de los tags crea la sintaxis del documento, concluimos que **los documentos XML se describen a sí mismos**.

Sintaxis XML

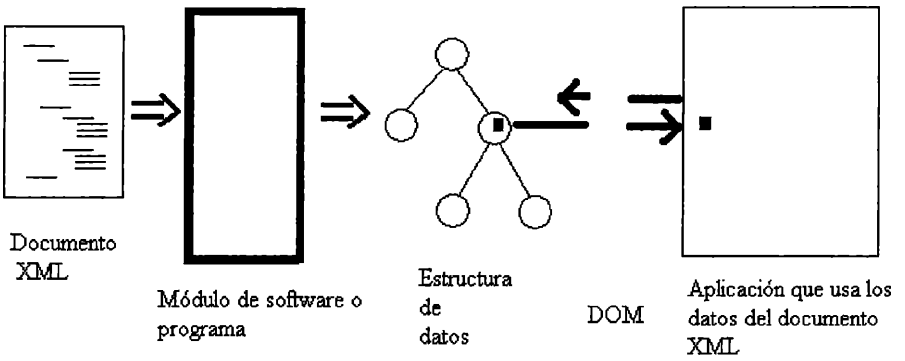
Veamos cómo se almacenan los datos y su estructura. La Recomendación XML 1.0 [W3CXML] ofrece todos los detalles, pero aquí veremos básicamente la sintaxis de XML o cómo escribir documentos XML bien formados.

La unidad central de información de XML es un **documento XML**, también llamado archivo XML (porque su extensión es xml), data packet o entidad XM.

Un documento XML bien formado tiene tres partes:

- *Prólogo* (opcional)
- *Cuerpo* del documento, con uno o más **elementos**. Los elementos deben estar organizados en forma de un árbol jerárquico. Un elemento puede contener **datos carácter** o bien cero, uno o más elementos.
- *Epílogo* (opcional)

Es importante tener en mente el papel que juega un documento XML en el sistema. La Recomendación XML 1.0 [W3CXML] asume que existe un módulo de software aparte que convierte el contenido físico del documento XML en una estructura de datos. La salida de este programa queda disponible para una aplicación del sistema (por ejemplo un Web browser). La interfase entre la representación interna de un documento creada por el módulo de software en cuestión y la aplicación, está especificada en una recomendación que se conoce con el nombre de DOM, por Document Object Model.



Ejemplo:

Si tenemos un documento XML, un programa crea a partir de él un árbol con la estructura de datos del documento. A partir de ése árbol, la aplicación que tiene que usar los datos del documento XML usa el DOM para extraer los datos.
Sea el documento XML

```
<referencias>
      <referencia>
      ...
      </referencia>
</referencias>
```

Primero se apunta a la raíz del árbol, formalmente, se crea una variable que haga referencia al objeto Documento.

Como tenemos un objeto Documento, su interfase expone métodos y propiedades que nos dicen cómo es el documento y nos permiten crear y buscar elementos.

Sea root la variable que apunta a la raíz. Entonces root.nodeName retorna el nombre del Nodo elemento raíz: "`referencias`"

Hay muchas propiedades y métodos del DOM que permiten manipular la estructura generada a partir del documento XML.

Por ejemplo un método del DOM es getFirstChild (Node n).

Otro es appendChild (nuevo_nodo), que agrega el nodo objeto nuevo_nodo a la lista de nodos hijos del nodo con cual se usa este método.

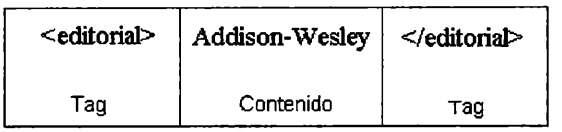
Existe otra interfase para manipular objetos de un documento XML llamada SAX. Esta interfase no crea la representación interna del documento. SAX no está desarrollada por la W3C y se usa cuando el costo de mantener el árbol que conforma la representación interna del documento es muy alto. Por ejemplo si el archivo XML es enorme.

Elementos

Los elementos son los que dan razón de ser al documento XML porque son los que efectivamente contienen los datos y su estructura. Un documento XML, tiene al menos un elemento.

Algunos elementos contienen texto y otros contienen otros elementos. La estructura de los elementos y los elementos en sí se describen mediante etiquetas o tags.

Un elemento es una entidad que comienza con un tag de inicio y termina con un tag de finalización. Entre ambos está el contenido del elemento.



Elemento

Ejemplo a:

```
<editorial>Addison-Wesley</editorial>
```

Si el elemento está vacío, entonces hay una notación especial. Es una manera abreviada de escribir los tags. En vez de escribir los tags de inicio y de finalización se escribe uno que termina con "/".

Ejemplo b:

```
<linea/>
```

XML es sensible a las mayúsculas. Por lo tanto <editorial> y <Editorial> son dos etiquetas distintas.

Atributos

Un atributo es un par nombre-valor que puede aparecer en la etiqueta de inicio de un elemento.

Ejemplo a:

```
<editorial categoría= "convenio">Addison-Wesley</editorial>
```

Ejemplo b:

```
<linea color="azul"/>
```

Un elemento puede tener cero, uno o más atributos. No importa el orden en que aparezcan (pero si un elemento tiene atributos, estos deben aparecer después del nombre de la etiqueta de inicio). Tampoco importa el tipo de comillas que se use, siempre y cuando el tipo de comillas que abre sea el mismo que el tipo que cierra.

Caracteres

XML usa los caracteres pertenecientes al conjunto de caracteres definidos en Unicode 2.1, congruente con ISO/IEC 10646, diseñado para incluir la mayor parte de los

caracteres internacionales. Es un estándar que para definir los caracteres usa 16 bits, no 7 como ASCII.

Unicode fue diseñado para ser un superset de la mayor parte de las codificaciones de caracteres, por lo tanto la conversión de ASCII a Unicode es casi directa.

Por ejemplo convertir ASCII a Unicode requiere dejar el byte menos significativo como está y completar el otro con ceros.

Una entidad parseada (un documento XML después de que un programa verificó que estaba bien formado) tiene texto, una secuencia de caracteres, el cual puede representar significado (markup) o datos carácter. Un carácter es una unidad atómica de texto como se especifica en ISO/IEC 10646.

Los caracteres permitidos incluyen tres caracteres de control ASCII C0 (Horizontal tab, Line-feed y carriage-return), todos los caracteres de display ASCII y casi todos los otros caracteres Unicode.

Nombres permitidos

Casi siempre las estructuras usadas en XML llevan un nombre. Los nombres XML pueden comenzar con una letra, guión bajo ("_", también conocido como underscore), o dos puntos (":", colon en inglés) y continúan con caracteres de nombre válidos.

El grupo de los caracteres de nombre válidos está formado por las letras, guión bajo, dos puntos (los mismos caracteres con los que puede comenzar un nombre XML) más los dígitos, guiones ("-", o hyphens) y los puntos. En la práctica los dos puntos no se deberían usar, excepto para los namespaces (una manera propia de XML de hacer que los nombres de distintos documentos no se confundan).

Como las letras permitidas no se limitan a los caracteres ASCII, se pueden crear etiquetas o tags a la medida de usuarios de todo el mundo, no sólo en inglés.

Una restricción que pesa sobre los nombres es que no pueden comenzar con la secuencia de caracteres "xml" ni con ninguna combinación de esta secuencia en mayúscula. No está permitido que los nombres comiencen con "XML", "xmL", etc. Los nombres que comienzan con esa secuencia están reservados para ser usados solamente por W3C.

Los tipos de datos principales

Los documentos XML están formados por caracteres o "character data" (CDATA). Esta CDATA se organiza en dos tipos:

- *CDATA que puede contener significado* (mediante markup, la manera de agregar significado a los datos) y por lo tanto necesita ser parseada (verificar que su estructura forme un árbol y que se cumplan las demás reglas, como por ejemplo que los nombres sean los permitidos. Esta data en formato carácter se llama PCDATA.

Por ejemplo los elementos son PCDATA, porque tienen una estructura que les da su significado.

```
<referencia>
      <identificador>Schwabe99</identificador>
      <autores>
            <autor>
                  <apellidos>
                        <apellido>Schwabe</apellido>
                  </apellidos>
                  <iniciales>
                        <inicial/>D
            </iniciales>
            </autor>
      </autores>
      <titulo>Improving Web information systems with navigational
patterns</titulo>
      <fuente>Computer Networks 31 pp.1667- 1678</fuente>
      <fecha>1999</fecha>
</referencia>
```

En este ejemplo el elemento `<referencia>...</referencia>` tiene su estructura en formato de árbol como exige XML. De esta manera el título "Improving Web information systems with navigational patterns" es más que una cadena de caracteres, ahora es el título de una referencia que se usa en un trabajo de grado. También es el texto del tercer hijo del objeto referencia.

- CDATA propiamente dicha, data en formato carácter que no contiene significado y por lo tanto no necesita ser parseada, (¿Con qué fin parsear algo que no tiene una estructura de la cual se pueda sacar provecho con XML?).
  Ejemplo:
  Supongamos que en el documento XML tenemos que almacenar este código: while (a<b), tenemos dos maneras.
  Una es usando la posibilidad de escapar el carácter "<". El resultado sería tener un elemento, por ejemplo mientras, de esta manera
  `<mientras>while (a>b)</mientras>`
  Otra posibilidad es usar el tipo CDATA. En este caso la solución sería
  `<mientras>  while(a b)  </mientras>`

Observación:
PCDATA es "data que va a ser parseada", no "data ya parseada". Por otra parte el parser o programa que realizará el parsing busca los caracteres "<" y ">" para identificar los inicios y finalización de las etiquetas, las cuales usa para verificar la estructura en forma de árbol del documento y así decir que el documento está bien formado. Entonces, ¿cómo almacenar en un documento XML *datos* que incluyan los caracteres "<" y ">"? La solución es algo llamado "escaping markup characters in PCDATA", ingresar marcas especiales para que el parser no asuma que "<" indica que a continuación viene una etiqueta.

DTDs y Schemas

Se llama lenguaje XML a un lenguaje que está definido en XML (porque XML es un meta lenguaje). Por tradición al lenguaje XML se lo llama "document type". Una gramática para ese lenguaje se llama "Document Type Definition", DTD.

La función de un DTD es limitar los tipos de elementos que se puede incluir en un documento. Define la gramática del documento.

Ejemplo

```
<!DOCTYPE referencia [
      <!ELEMENT referencia (identificador, autores, titulo, editorial,
fecha)>
      <!ELEMENT identificador (#PCDATA)
      <!ELEMENT autores (autor +) --uno o más autores-->
      <!ELEMENT autor (apellidos, iniciales)
      <!ELEMENT apellidos (apellido +)
      <!ELEMENT apellido (#PCDATA)
      <!ELEMENT iniciales (inicial +)
      <!ELEMENT inicial (#PCDATA)
      <!ELEMENT titulo (#PCDATA)
      <!ELEMENT editorial (#PCDATA)
      <!ELEMENT fecha (#PCDATA)
]>
```

Es el DTD correspondiente a

```
<referencia>
      <identificador/>
      <autores>
            <autor>
                  <apellidos>
                        <apellido/>
                  </apellidos>
                  <iniciales>
                        <inicial/>
                  </iniciales>
            </autor>
      </autores>
      <titulo/>
      <editorial/>
      <fecha/>
</referencia>
```

El DTD no es obligatorio. Además puede aparecer dentro del documento XML o aparte. En este último caso en el documento XML se agrega una referencia al DTD. Tiene el inconveniente de que sigue una sintaxis diferente de la que siguen los documentos XML y de que no distingue tipos de datos. El uso de DTD está regido por la Recomendación XML 1.0.

Como resultado de las limitaciones de los DTD, la W3C está trabajando en propuestas para definir y manipular la estructura de un documento XML y darle (aún siendo sólo texto) la capacidad de tener tipos de datos.

Actualmente el Working Group de Schemas XML está trabajando en dos especificaciones para schemas. Una trata las facilidades para controlar y describir las reglas estructurales de un documento, [W3CXMLSCHEMA1]. La otra trata el tema de la definición de tipos de datos para los items que forman parte del contenido [W3CXMLSCHEMA2].

Los schemas son la parte de la familia XML que muy probablemente reemplazará a los DTD. Hoy en día no se usan porque, al estar la W3C trabajando, las propuestas pueden cambiar, pero conviene tenerlos en cuenta porque en el futuro habrá recomendaciones para usarlos y como a la hora de programar el hecho de poder asignarle tipo a los datos simplifica la tarea, es de esperar que sean muy usados. Por eso los mencionamos aquí.

La idea general de un schema XML es que sea un documento que describa y restrinja un conjunto de instancias de documentos XML. Un schema XML es él mismo un documento XML.

Esto permite que una aplicación XML pueda obtener información del schema con facilidad. También permite al diseñador incorporar al schema información para ser usada sólo por la aplicación.

Por ejemplo, el diseñador podría incorporar información a la definición de cada elemento conteniendo su clasificación privada. Luego podría usar esta información privada en la aplicación para decidir qué datos mostrar a qué usuario.

Según la W3C el propósito de un schema es "definir y describir una clase de documentos XML usando los constructores de significado [markup] para restringir y documentar el significado, uso y relaciones de las partes que lo componen: tipo de datos, elementos y su contenido, atributos y su contenido, entidades y su contenido"..."Los constructores schema pueden también proveer a la especificación de información adicional como valores por defecto"..."las estructuras schema de XML se pueden usar para definir, describir y catalogar los vocabularios XML para clases de documentos XML".

Ejemplo:
Consideremos el siguiente DTD,

```
<!ELEMENT referencia (identificador, autor, titulo, editorial?)>
<!ELEMENT identificador (#PCDATA)
<!ELEMENT autor (apellido, inicial)
<!ELEMENT apellido (#PCDATA)
<!ELEMENT inicial (#PCDATA)
<!ELEMENT titulo (#PCDATA)
<!ELEMENT editorial (#PCDATA)
```

El schema correspondiente sería,

```
<Schema...>
      <element name="Referencia">
            <type>
                  <element name="identificador"
                           type="string"/>
                  <element name="autor">
                        <type>
                              <element name="apellido"
                                       type="string"/>
                              <element name="inicial"
                                       type="string"/>
                        </type>
```

```
                    </element>
                    <element name="titulo"
                              type="string/>
                    <element name="editorial"
                              type="string "   minOccurs="0"
                    maxOccurs="1"/>
              </type>
      </element>
</Schema>
```

Documentos bien formados y documentos válidos

Un documento está **bien formado** si respeta las reglas de formato de documentos que aparecen en la recomendación de XML. Principalmente esto significa que debe tener un único elemento raíz y que todos los demás elementos deben ser sus hijos o descendientes más lejanos y aparecer adecuadamente anidados formando un árbol a partir de la raíz. Si un documento está bien formado, entonces puede ser parseado por un programa. Si no está bien formado, el documento no tiene sentido como XML y cualquier programa que trabaje con él debe reportar un error y debe interrumpir el procesamiento del documento excepto para reportar más errores.

Ejemplo a: Este es un documento bien formado.

```
<saludo>hola</saludo>
```

Ejemplo b: Este no es un documento bien formado

```
<saludo>hola
```

Ejemplo c:

```
<?xml version="1.0" encoding="iso-8859-1"?>
      <referencia>
            <!--libro-->
            <identificador>Nielsen99</identificador>
            <autores>
                  <autor>
                        <apellidos>
                              <apellido>Nielsen</apellido>
                        </apellidos>
                        <iniciales>
                              <inicial>J</inicial>
                        </iniciales>
                  </autor>
            </autores>
            <titulo>Designing Web Usability: The Practice of
Simplicity</titulo>
            <editorial>New Riders Publishing</editorial>
            <fecha>1999</fecha>
      </referencia>
```

Este documento XML está bien formado. La raíz es `<referencia>`, `<?xml version="1.0" encoding="iso-8859-1"?>` es la declaración de XML, pertenece al prólogo, es opcional, pero conveniente.

Los documentos que cumplen las reglas definidas en un DTD son documentos **válidos**.

Estructura lógica de un documento XML

Desde el punto de vista lógico, un documento XML puede tener los siguientes tipos de componentes:

Tipo de componente	Delimitado por
Declaraciones en el DTD	<!DOCTYPE[...]> <!ELEMENT...> <!ATTLIST...> <!ENTITY...> <!NOTATION...>
Elementos	<etiqueta...>...</etiqueta> o </etiqueta>
Secciones CDATA	<![CDATA[...]]>
Comentarios	<!-- . . . -->
Instrucciones de Procesamiento (PIs)	<? . . . ?>

Procesador

Un procesador es un programa que manipula documentos XML de tal manera que recibe el documento XML y obtiene una estructura de árbol que puede ser manipulada mediante scripts o programas.

La W3C define procesador XML como un módulo de software "usado para leer documentos XML y proveer acceso a su contenido y estructura". [W3CXML]

"Se asume que un procesador XML trabaja en beneficio de otro módulo, llamado aplicación". La especificación XML [W3CXML] "describe el comportamiento requerido de un procesador XML en términos de cómo debe leer la data XML y la información que debe brindarle a la aplicación."

Muchas veces se usa el término parser para designar la misma entidad que la W3C denomina procesador, lo mismo ocurre en este trabajo, donde usamos los dos términos.

Así como hay documentos bien formados y documentos válidos, hay parsers que validan y parsers que no validan.

Un parser que no valida crea la estructura de árbol a partir de un documento XML sin tener en cuenta las restricciones de ningún DTD. Un parser que valida verifica que los documentos cumplan con las restricciones de su DTD, si tiene uno. Si el documento no tiene un DTD, el parser que valida actúa como uno que no valida. Un ejemplo de un parser que valida es el parser de Microsoft , MSXML, incluido en IE5.

Además existe una especificación denominada XML Information Set o Infoset que el procesador usa como "contexo" de un documento XML [W3CInfoset]. Consiste en quince tipos de información que conforman un documento bien formado. Todo procesador de XML retorna el contenido del documento XML. Ese contenido se describe en términos de los "*information types*" de Infoset.

Una representación parseada de un documento XML que sea congruente con el documento original tdebe tener algunos items pertenecientes a alguno de estos information types y puede no tener otros.

Existen information types *requeridos* y *opcionales*. La representación parseada de un documento que sea congruente con el documento XML original debe tener los items requeridos y puede no tener los opcionales.

Instrucciones de procesamiento

Son maneras de invocar alguna aplicación que no sea el parser para realizar algún procesamiento sobre una parte del documento XML.

El formato que siguen es:

```
<?nombreAplicacionQueRealizaProceso  Conjunto de instrucciones
para la aplicación?>
```

Una desventaja de las instrucciones de procesamiento es que no existe conexión entre el contenido del documento XML y las PI (Instrucciones de Procesamiento). Las PIs no se pueden anidar dentro de un elemento particular. Por otra parte no son elementos XML y no usan la sintaxis estándar.

No obstante, una importante ventaja es que sirven para incorporar comportamiento externo.

Una Instrucción de Procesamiento es un ítem en un documento XML que por convención se usa para darle instrucciones al software que recibe el documento y lo procesa. Para detectarlo se escribe entre "<?" y "?>".

Observación: la declaración XML en el prólogo del documento no es una instrucción de procesamiento, aún cuando use los mismos delimitadores. (Ejemplo de declaración: `<?xml version="1.0"?>`)

Se usan cuando se desea que la aplicación que está realizando el procesamiento del documento XML lleve a cabo alguna acción cuando alcanza un punto determinado del documento. En ese caso embebemos una Instrucción de Procesamiento para indicar que alguna acción debe ocurrir en ese lugar (donde se encuentra la PI).

El parser indica la presencia de una PI mostrando un nodo de Instrucción de Procesamiento en el lugar adecuado en el árbol de nodos (esto si es un parser que primero arme el árbol de nodos, si es un parser que no lo arma, entonces dispara un evento de Instrucción de Procesamiento).

El código que conduce al programa que está procesando el documento, ejecuta alguna acción sobre el documento de acuerdo a lo que indique la PI.

Ejemplo

```
<?xml version="1.0"?>
<referencias>
   <referencia>
      <?archivo 128>
      <identificador>XMLPAT
      </identificador>
      <autores>
         <autor>
            <apellidos>
               <apellido/>
            </apellidos>
            <iniciales>
               <inicial/>

            </iniciales>
                  </autor>
            </autores>
            <tituloXML
Patterns>

<fuente>http://www.x
mlpatterns.com
            </fuente>
```

```
    <fecha>2000</fecha>
  </referencia>

<!-- más referencias pueden ir
acá-->

</referencias>
```

En el ejemplo el string <?archivo 128> es la declaración de Instrucción de Procesamiento. El string archivo identifica la "aplicación" que realiza el procesamiento (el "target" u objetivo de la PI).

El string 128 es el conjunto de instrucciones para la aplicación, el pseudo-atributo que acompaña la instrucción.

Las Instrucciones de Procesamiento siempre tienen dos partes: la aplicación destino (o processing instruction target) y la información que se le pasa. Todo lo que aparece antes del espacio en blanco es "processing instruction target y lo que aparece después del blanco es información que se pasa.

La declaración de Instrucción de Procesamiento en el ejemplo indica que la información de la referencia es la más actual y que si se modifica, entonces hay que almacenarla en un archivo de referencias viejas bajo el número 128.

Al igual que los tags de XML, las Instrucciones de Procesamiento no pueden comenzar con el string XML en cualquiera de sus variantes de mayúsculas y minúsculas porque los nombres que comiencen así están reservados para ser usados por la W3C.

Existe una instrucción de procesamiento muy útil porque sirve tanto para seleccionar la stylesheet con la cual mostrar la información del documento como para desarrollar transformaciones de datos. Es la Instrucción de Procesamiento `<?xml-stylesheet?>`. Esta instrucción se usa dentro de un documento XML fuente para identificar la stylesheet que debería usarse para procesarlo. Puede haber varias instrucciones `<?xml-stylesheet?>` presentes, definiendo diversas stylesheets para ser usar bajo distintas circunstancias.

Esta Instrucción de Procesamiento tiene un pseudo-atributo *href* cuyo valor es el URI (Uniform Resource Locator) de la stylesheet y un pseudo-atributo type que indica el lenguaje en el cual está escrita la stylesheet. Type puede ser *text/xml* o bien *application/xml*. Sin embargo, con la implementación de Microsoft en Internet Explorer 5 debe ser *text/xsl*.
La Instrucción de Procesamiento <? xml-stylesheet?> debe aparecer, si lo hace, en el prólogo del documento.

Ejemplo

```
<?xml-stylesheet type="text/xsl" href="../mistylesheet.xsl"?>
```

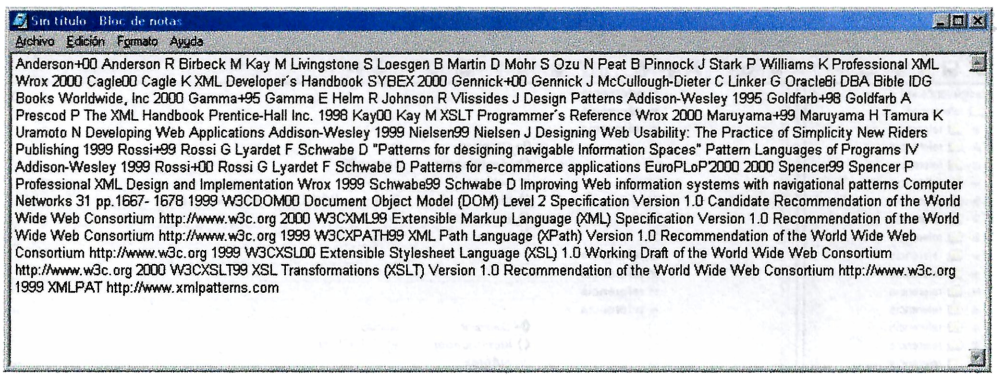
Una de las razones para separar el la stylesheet del documento XML fuente es permitir que la misma información se presente de diversas maneras al usuario, dependiendo del usuario, del equipo que use, etc.

Un ejemplo real

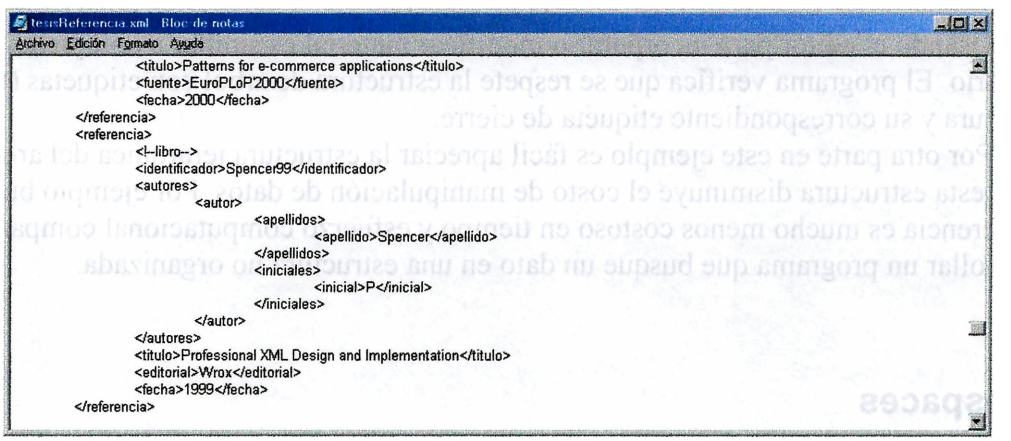
Lo importante es que usando XML el programador recibe una jerarquía de objetos. No tiene que parsear. No tiene que programar que del carácter 1 al 10 es el código de cliente, etc.

La bibliografía de este Trabajo de Grado se creó en un archivo XML denominado TesisReferencia.xml. Se encuentra una parte de este archivo en la sección "Referencias de la Tesis como figura en los ejemplos" del Apéndice. Para editarla con el objeto de incorporar material se usó el bloc de notas, dado que un archivo xml no es más que un archivo de texto común.

La bibliografía sin caracteres que agreguen espacio y usando el bloc de notas se observa así:



Parte de la misma bibliografía almacenada en un archivo XML (TesisReferencia.xml) con caracteres de espacios se observa así:



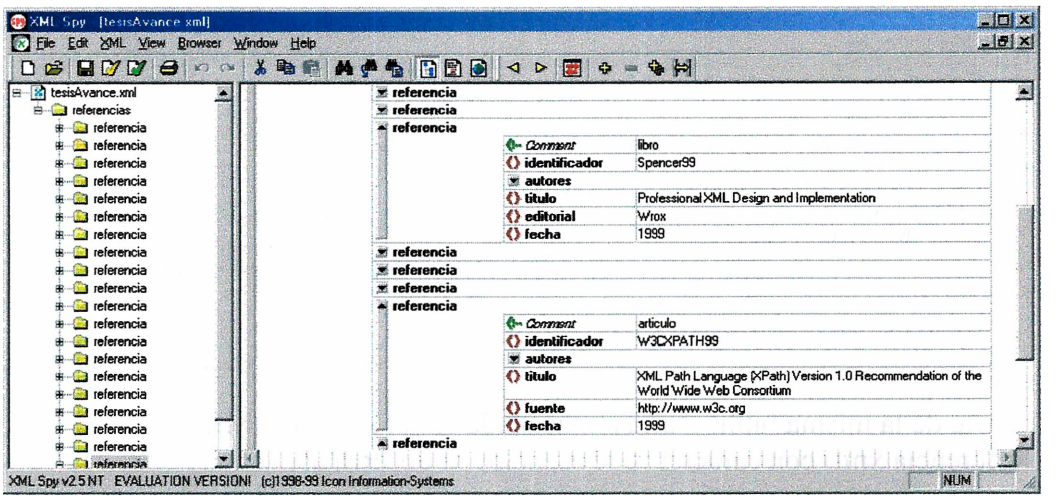
Aún sin usar un editor específico de XML, se observa la estructura de árbol jerárquico implícita.

Pero además de editar el archivo XML con el Block de notas, también se lo puede hacer con un editor de XML. El editor de xml usado en este ejemplo se llama XML Spy (http://www.xmlspy.com). Un editor de XML es un programa que permite ver y modificar un archivo XML.

La ventaja de usarlo radica en que muestra los datos respetando la estructura de árbol que impone XML. En general los editores también realizan tareas de validación y

verificación, de hecho XML Spy las realiza. Podemos pensar en las tareas de validación y verificación como las tareas análogas a las de verificar la ortografía y gramática que realiza un procesador de texto (con la diferencia de que un procesador de XML no tiene permitido continuar la ejecución, salvo para reportar errores, una vez que encuentra un error de verificación). Cuando existe un error tal que el archivo XML está mal formado, el programador debe corregirlo. En estos casos el editor de XML es muy útil porque muestra la información de manera jerárquica y, sobre todo, revisa la sintaxis e informa dónde está el error. Si consideramos que nada limita el tamaño de los archivos XML, la revisión y edición disminuyen drásticamente los tiempos de corrección de errores.

Vista con XML Spy, un editor de XML, parte de la bibliografía se observa así:



Usando el editor para incorporar o modificar material es similar a completar un formulario. El programa verifica que se respete la estructura de árbol con etiquetas (tags) de apertura y su correspondiente etiqueta de cierre.

Por otra parte en este ejemplo es fácil apreciar la estructura jerárquica del archivo y cómo esta estructura disminuye el costo de manipulación de datos. Por ejemplo buscar una referencia es mucho menos costoso en tiempo y esfuerzo computacional comparado a desarrollar un programa que busque un dato en una estructura no organizada.

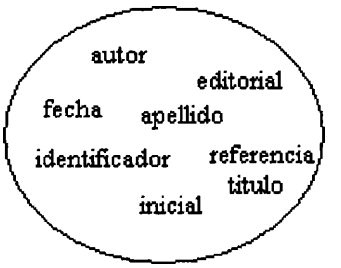
Namespaces

Qué es un namespace

Es un espacio de nombres al cual se le asigna un **identificador único**.

Los namespaces son las herramientas que provee XML para definir el alcance de las entidades y los atributos.

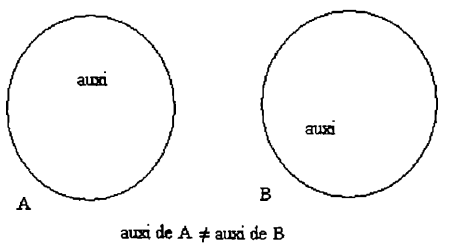
XML permite mezclar distintos vocabularios en un mismo documento XML y hay muchos usos potenciales para los elementos, por lo tanto se necesita una manera de identificar el contexto de un elemento. El namespace define el contexto del elemento. La identificación del contexto se hace mediante los namespaces y para su uso existe una especificación. La especificación de namespaces se encuentra en [W3Cnamespaces].



Referencias de una Tesis
(conjunto de algunos elementos que aparecen en la Tesis)

Objetivo

El objetivo es usar tags de dos vocabularios distintos en un mismo documento sin que esto genere conflicto de nombres. [Kay00]



auxi de A ≠ auxi de B

De esta manera en el ejemplo de la figura auxi de A no se confunde con auxi de B.

XML brinda la posibilidad de que la gente cree sus propios tags. En consecuencia es muy probable que la gente use el mismo nombre de elemento para cosas distintas. Y aunque el significado fuese el mismo, el contenido de los elementos podría variar entre las distintas definiciones.

Por ejemplo supongamos que tenemos dos aplicaciones. Una aplicación organiza la comunicación entre los trabajadores de una empresa, mientras que otra liquida sus sueldos. El `<empleado>` perteneciente al vocabulario de una aplicación no es lo mismo que el `<empleado>` del vocabulario de la otra aplicación. La ventaja de esta separación consiste en que ambas aplicaciones pueden usar `<empleado>` definiéndolo de manera independiente sin ocuparse de cómo lo define la otra. Si estas aplicaciones tuvieran que interactuar, aún así no habría confusión entre los dos elementos ya que cada elemento empleado estaría "calificado" con el namespace que le correspondiera y por lo tanto cada aplicación podría identificar el elemento empleado que necesita.

Por ejemplo la aplicación A define que `<empleado>` tiene tres elementos: `<mail>`, `<nombre>` y `<telefono>` mientras que para la aplicación B, que liquida sueldos, `<empleado>` tiene 25 elementos (`<nombre>`,`<apellido>`,`<CUIT>`,`<ingreso>`,`<estado_civil>`,`<puesto>`,etc).

Qué se usa como identificador único

Como identificador único se usa el URI, Unique Resource Identifier.
El URI puede tomar varias formas. Una de estas formas es la conocida URL Uniform Resource Locator. Por ejemplo un identificador único es http://www.unlp.edu.ar

Lo importante del URI es que sea único.

En la práctica conviene usar como identificador del espacio de nombres (namespace) una URL que los demás no usen. Por esta circunstancia se usa la URL del sitio Web que se posea.

Por ejemplo si la empresa ACME tiene su sitio en www.acme.com, entonces es útil definir el namespace como www.acme.com/namespaces.

No es necesario que exista el dominio ni ningún elemento al cual apunte el URL que uso como namespace, lo importante es que sea único y definiéndolo de esta manera hay pocas posibilidades que exista en un documento XML ajeno a ACME con el identificador de ACME y un nombre de elemento usado por ACME.

Prefijos

Definido como se sugiere en el punto anterior, el URI tiende a ser un nombre largo y con barras "/". Por lo tanto resulta incómodo para usar. En consecuencia se crea en el documento XML un sobrenombre (nickname) para usar el namespace. Este sobrenombre se usa como prefijo de los nombres de elementos y atributos en el documento.

De esta manera un elemento o atributo tienen un nombre *real* conformado por el prefijo (que se asocia al namespace en el documento) más el nombre *local*.

Nombre real del elemento o atributo = namespace + nombre local Namespace->Prefijo => Nombre real del elemento o atributo = prefijo + nombre local

Ejemplo
Nombre del elemento = template
Identificador de namespace= `http://www.w3.org/1999/XSL/Transform`

- ⇨ Prefijo= `xsl`
- ⇨ Como prefijo = identificador de namespace,
- ⇨ El nombre del elemento template es

```
<xsl:template
...>
  contenido del elemento
  ...
</xsl:template>
```

Y cuando se encuentra `<xsl:template ...>` en el documento se identifica a *este* elemento template y no a otro.

El elemento de nombre template con el namespace que aparece en el ejemplo es un elemento de XSLT y se usa en stylesheets para generar una salida.

Observación: Como se describe en la sección de XSL, las stylesheets son documentos xml que tienen instrucciones para definir la salida de documentos xml.

Si bien existe el elemento template de XSLT cuya función es generar una salida, también se podría definir otro elemento template que no tuviera nada en común con este.

Por ejemplo podríamos estar trabajando con una Base de Datos de templates de documentos Word. En este caso definiríamos un elemento template con la información necesaria para nosotros.

Los dos templates no se relacionan y pertenecen a vocabularios distintos, sin embargo al momento de procesar los archivos xml hay que trabajar con cada uno según el caso. ¿Cómo se distingue a qué contexto pertenece el elemento template? La respuesta es por el namespace que figure en su nombre.

Ejemplo

Elemento XSLT	template

Template es un elemento cuya función es generar una salida

Namespace	`http://www.w3.org/1999/XSL/Transform`

Para abreviar, declaramos el prefijo y elijo por su nombre xsl.

```
xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
```

Entonces el formato del elemento template es:

```
<xsl:template
            name=Qname
            match=Pattern
            mode=Qname
            priority=Number>
                  <xsl:param>*
                  cuerpo del template
                        ...
</xsl:template>
```

Template es un elemento de la base de datos de templates de la empresa ACME, en otro vocabulario y en otro entorno de trabajo. Entonces podriamos usar un namespace para identificarlo y no confundirlo.

Sea el namespace http://www.acme.com/namespacesproyxml, y definimos el prefijo docu, así:

```
xmlns:docu = "http://www.acme/namespacesproyxml"
```

Ahora el formato de este elemento template es:

```
<docul:template
            name=Qname>
                  <font/>
                  <tipo/>
                  ...
                  otros subelementos
</docu:template>
```

Si los dos elementos tuviesen que aparecer en un mismo documento xml, entonces no habría confusión porque uno se identifica con xsl:template y el otro con docu:template.

Observación: si hubiésemos elegido como nombre del prefijo xsl y no se mezclasen los dos elementos en un mismo documento xml, no hubiese habido problema. Porque el procesador asocia el prefijo con el namespace con el cual se lo definió.

Explicación del ejemplo

Tenemos una base de datos de templates de Word por un lado y el elemento template que pertenece al conjunto de elementos XSLT, estos últimos interpretados por programas especiales.

Un procesador XSL es en esencia un programa que lee instrucciones y genera una salida (para más detalles ver XSLT).

Cuando un procesador XSLT encuentra un elemento template cuyo identificador de namespace (es decir, namespace) es http://www.w3.org/1999/XSL/Transform, entiende que es un elemento template que pertenece al conjunto de los elementos de XSLT y genera una salida de datos.

Cuando encuentra el elemento template perteneciente a la base de datos de templates, lo trata como un dato.

Cómo se declara el prefijo de un namespace

El prefijo de un namespace se declara usando el pseudoatributo especial **xmlns**. Ejemplo

```
xmlns:xsl = http://www.w3.org/1999/XSL/Transform
```

Aquí xmlns es el pseudoatributo,

`http://www.w3.org/1999/XSL/Transform` es el identificador único del namespace y `xsl` es el prefijo.

Por definición, xmlns se puede usar dentro de cualquier elemento. También podemos usar el prefijo declarado en los nombres de elementos, nombres de atributos contenidos.

Los elementos que no tienen prefijo pueden usar el namespace por defecto. El namespace por defecto se declara también usando el pseudoatributo xmlns, pero de esta forma:

```
xmlns="URI como identificador de namespace"
```

XLINK

Es la especificación dedicada a describir una manera estándar de incorporar "hyperlinks" (links de hipertexto) a un archivo XML. Se la encuentra en [W3CXLink]. "Define un lenguaje denominado XLink, por XML Linking Language, el cual permite insertar en documentos XML elementos que sirven para crear y describir links entre recursos." [W3CXLink]

Las estructuras que describen los links respetan la sintaxis XML.

Los links son parecidos a los clásicos links que se usan en HTML. Usando XLink se pueden crear links simples como los de HTML, pero también algunos más complejos.

Los links que siguen la especificación XLink pueden:

- Establecer relaciones entre dos o *más* recursos. Por ejemplo uno de los usos comunes de XLink es crear hyperlinks, peor también podría crear un links desde un recurso a varios.
- Relacionar mediante el link datos sobre los cuales *no tenemos permisos de escritura.* Por ejemplo podríamos tener links con comentarios a documentos de otros.
- Mantener links que residen en una ubicación diferente de los recursos conectados. Por ejemplo podríamos almacenar los links en una base de datos.

Por definición un link es una relación explícita entre recursos o partes de recursos. Un recurso es cualquier unidad de información o servicio direccionable.

Como los XLinks respetan la sintaxis XML, los XLink son elementos insertados en los documentos XML. Por otra parte en [W3CXLINK] se define una aplicación XLink como "cualquier módulo de software que es capaz de interpretar documentos XML bien formados que contengan elementos y atributos XLink...".

Los elementos, para ser considerados XLink, deben cumplir las restricciones especificadas en [W3CXLINK] (debe tener un atributo "type" que pertenezca al namespace de XLink. El valor de este atributo puede ser "simple", "extended", "locator", "arc", "resource", "title", o "none" teniendo cada uno de estos valores un significado especial para el programa encargado de interpretarlos como indica la especificación de XLink).

Para que las aplicaciones puedan manipular y atravesar un XLink, deben reconocer que están frente a un elemento XLink. Con el fin de reconocer constructores pertenecientes al vocabulario XLink, en la especificación de XLink se definió un namespace XLink. Este namespace tiene forma de URI y es `http://www.w3.org/1999/xlink`.

Ejemplo a:

En un documento XML se ingresa la declaración
`xmlns:xlink="http://www.w3.org/1999/xlink` que hará que el prefijo xlink esté disponible dentro del elemento miElemento con el objeto de representar el namespace XLink. Este ejemplo pertenece a [W3CXLINK]

```
< myElement
  xmlns:xlink="http://www.w3.org/1999/xlink">
  ...
</myElement>
```

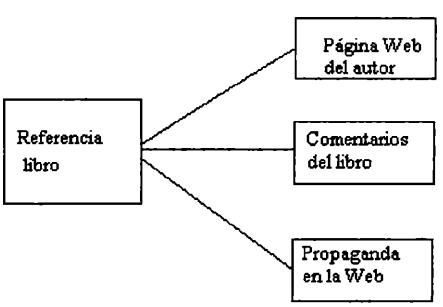
Ejemplo b:

En un documento XML donde almacenamos las referencias, se ingresa el elemento XLink que apunta a la fuente de una referencia.

```
<referencias>
  xmlns:xlink="http://www.w3.org/1999/xlink">
<!-- más elementos referencia aquí-->
      <referencia>
            <identificador>TECHWEB</identificador>
            <autores>
                  <autor>
                        <apellidos>
                              <apellido>Gonsalves</apellido>
                        </apellidos>
                        <iniciales>
                              <inicial>A</inicial>
                        </iniciales>
                  </autor>
            </autores>
            <titulo>Vendedores importantes lanzan iniciativa B-to-
B</titulo>
            <fuente
                  xlink:type="simple"
                  xlink:href=http://www.techweb.com/news>
            Techweb
            </fuente>
            <fecha>25/10/2000</fecha>
      </referencia>
</referencias>
```

El *atributo* type indica el tipo de elemento XLink (simple, extended, locator, arc, resource, o title); a su vez, el *tipo de elemento* indica las restricciones que XLink impone al elemento XLink (por ser un elemento de *tal* tipo) y el comportamiento que deben tener las aplicaciones XLink cuando encuentren este elemento.

En el ejemplo el atributo type tiene el valor "simple", entonces el elemento fuente es de tipo simple, entonces debe cumplir las reglas de XLink para elementos simples y cualquier aplicación que se tope con este elemento actúa sobre él con el comportamiento correspondiente a elementos simples.



Un tipo de link extended es un link que asocia un número arbitrario de recursos. Por ejemplo una referencia de un libro podría

señalar en el mismo link varios puntos de contacto con la fuente, (una lista de links relacionada con el recurso) como la página Web del autor, un artículo a cerca del libro o la página de propaganda del libro en una tienda virtual.

XPATH

XPath es el nombre de XML Path Language, un mecanismo para consultar estructuras XML.Es un lenguaje sofisticado para desarrollar todo tipo de búsquedas en un documento. Es muy usado en XSL (otro miembro de la familia XML). [Cagle00] También provee una manera universal de especificar alguna porción de un documento XML.

Una expresión XPath se puede usar para realizar cálculos, manipular strings o verificar condiciones booleanas, pero su uso más característico es identificar partes del documento XML de entrada para ser procesadas.

El mecanismo para extraer información de XML debe involucrar el menor esfuerzo posible tanto para el programador como para el parser. Para busca los datos hay que navegar el árbol que se generó a partir de la estructura XML. Se necesita una manera eficiente de extraer datos de ese árbol y un lenguaje que permita navegar y desarrollar todo tipo de búsquedas en el árbol XML.

Ejemplo

Tenemos el archivo de referencias que figuran en la Tesis y deseamos mostrarle al usuario una lista con los elementos de las referencias cuya fuente es el World Wide Web Consortium. Los elementos fuente de las referencias que se busca tienen la dirección http://www.w3.org, es su característica.

Entonces la tarea se reduce a transformar el documento para seleccionar los datos buscados y darle formato de página HTML con una tabla.

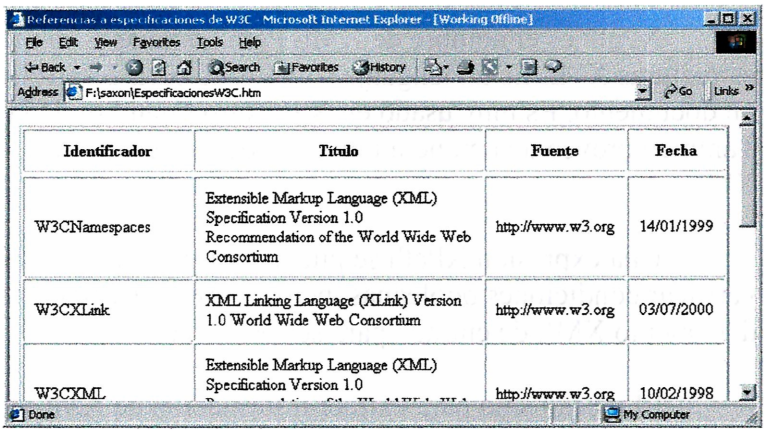
El procesador genera una estructura con forma de árbol y es ése arbol el que consulta usando XPath para encontrar la información.

El atributo select contiene una expresión XPath que busca los elementos referencia que tengan un elemento fuente hijo cuyo valor sea http://www.w3.org.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet
      version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
      <html>
      <head>
            <title>Referencias a especificaciones de W3C</title>
      </head>
      <body>
  <table border="2" cellpadding="10">
   <tr>
<th>Identificador</th><th>Título</th><th>Fuente</th><th>Fecha</th
>
   </tr>
```

```
        <tr>

      <xsl:for-each
select="//referencia[fuente='http://www.w3.org']">
          <tr>
              <td><xsl:value-of select="identificador"/></td>
              <td><xsl:value-of select="titulo"/></td>
              <td><xsl:value-of select="fuente"/></td>
              <td><xsl:value-of select="fecha"/></td>
          </tr>
      </xsl:for-each>
          </tr>
      </table>
          </body>
          </html>
</xsl:template>
</xsl:stylesheet
>
```



Una vez que el documento XML tesisReferencia.xml es procesado por un procesador (en este caso se usó el procesador saxon) que recibe instrucciones de la stylesheet imprimirAlgunos.xsl (el programa que aparece arriba), generamos un archivo htm. En este caso el archivo generado se llama EspecificacionesW3C.htm y un browser lo muestra como aparece en la figura de abajo.

XPOINTER

La recomendación XPointer está dedicada a señalar partes de un documento XML. Se la encuentra en [W3CXPointer].
Un XPointer es parecido a un URL, pero en lugar de apuntar a documentos en la Web, apunta a datos dentro de un archivo XML.

Por ejemplo con XPointer podemos apuntar a subárboles o atributos de un documento XML. De esta manera el procesador puede mostrar la parte apuntada solamente, sin necesidad de mostrar toda la página.

XPointer está basado en XPath, soporta el direccionamiento a estructuras internas de los documentos XML. Permite recorrer el árbol del documento y seleccionar partes internas en base a varias propiedades, como por ejemplo el tipo de los elementos, los valores de los atributos, el contenido carácter y la posición relativa.
Al momento de escribir esta tesis aún está cambiando esta especificación. No obstante más información se encuentra en [W3CXPointer].

XSL

XSL es un lenguaje para escribir stylesheets. Tiene dos partes [W3CXSL]:

- XSL Transformations (XSLT) un lenguaje para transformar documentos XML
- Un vocabulario XML para especificar la semántica de formato (XSL Formatting Objects)

Junto con XPath, es la especificación que se encarga de la extracción de información.

Orígenes XSL

XSL, el acrónimo de eXtensible Stylesheet Language fue creado para "definir el formato y la representación de documentos XML para mostrarlos por pantalla, papel u oralmente."[Kay00]

Mientras se desarrollaba XSL se hizo aparente que para mostrar XML había que hacer dos cosas: una era seleccionar elementos, agruparlos y reordenarlos, otra era darles formato para que los elementos resultantes fuesen mostrados como texto, pixels, etc y de la manera deseada.

Por lo tanto se separó XSL en dos: XSLT para definir las transformaciones y XSL para dar formato a los resultados.

XSLT

XSLT es el acrónimo de eXtensible Stylesheet Language Transformations. Es un lenguaje diseñado para transformar un documento XML. Pertenece a la familia de estándares XML

XSLT puede transformar un documento XML en otro, XML en HTML y XML en otros formatos basados en texto. La especificación de XSLT se encuentra en [W3CXSLT].

Existen procesadores XSLT que son programas que respetan las especificaciones para procesar el árbol XML de los documentos y generan salidas que también tienen formato según las especificaciones.

Ejemplos de procesadores XSLT son

- Instant Saxon (http://users.iclway.co.uk/mhkay/saxon/instant.html),
- msxml3 de Microsoft y
- xt de J. Clark.

¿Qué es transformar un documento XML?

Es recibir un documento XML con una estructura y devolver otro documento que puede ser XML con otra estructura, texto, html, etc.

¿Por qué se necesita transformar XML?

XML cumple dos objetivos:
1) separar los datos de la representación
2) transmitir datos entre aplicaciones

XML es un mecanismo que **unifica las formas de representar estructuras.** Esto significa que las estructuras se almacenan sin tener en cuenta si van a ser vistas por una persona o consumidas por una aplicación. Por lo tanto **la data XML en general no se utiliza en el formato en el que se almacena**, sino que es necesario transformarla para poder utilizarla.

Ejemplo a:
Para que una persona pueda utilizar datos es necesario mostrárselos por pantalla, imprimirlos o leerlos.
Es decir que, por ejemplo, habrá que mostrar los datos en un browser, y para esto habrá que obtener los datos primero y generar un archivo html. Es decir: convertir XML a HTML.

El uso más común hoy de XSLT es convertir XML a HTML.

Ejemplo b:
Para transferir data de una aplicación a otra se necesita transformar la data del modelo de datos que usa una aplicación al modelo de datos que usa la otra.
Por ejemplo una aplicación puede almacenar los datos en una estructura XML y otra necesitar recibirlos como un script SQL:
Ejemplo b1)

Podría almacenar la consulta así:

```
<consulta>
<tipo>select</tipo>
<tablas>
<tabla>tarjetas_credito</tabla>
</tablas>
<columnas>
<columna>saldo
</columna>
</columnas>
<filtro>
<condicion>
<columna>tipo</columna>
<valor>Visa</valor>
</condicion>
<condicion>
<columna>numero</columna>
<valor> 1234567891234567</valor>
</condicion>
</filtro>
</consulta>
```

La transformación recibe los datos de la primera aplicación y genera y retorna el script que necesita la segunda.

```
select saldo from tarjetas_credito where tipo='Visa' and numero =
1234567891234567
```

Ejemplo b2)

En este ejemplo una aplicación tiene los datos respetando la estructura de árbol de XML. Un procesador recibe el árbol y devuelve texto plano para una segunda aplicación que necesita los datos en formato texto separados por comas.

```
<cliente>
<nombre>Adam Scott</nombre>
<domicilio>14 N 1566</domicilio>
<profesion>analista de sistemas</profesion>
<login>adam</login>
<password>amigo</password>
</cliente>

Adam Scott, 14 N 1566, analista de sistemas, adam, amigo
```

Ejemplo c:

Dos archivos XML pueden tener distinto vocabulario aunque estén almacenando datos semánticamente iguales:

Supongamos que Adam Scott trabaja en una empresa y que esta empresa paga los gastos de su celular.

Compañía Telefónica 1:

```
<cliente>
<nombre>Adam Scott</nombre>
<plan>premium</plan>
...
<minutos_aire>129</minutos_aire>
<monto>235</monto>
</cliente>
```

Empresa que abona las llamadas del celular de su empleado:

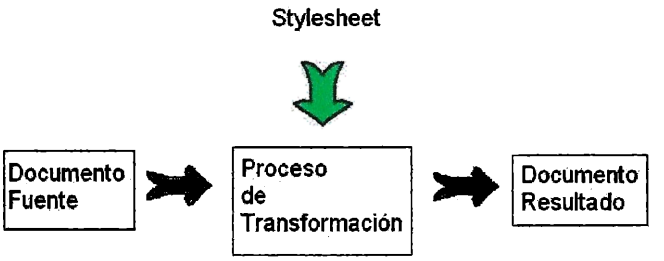
```
<empleado>Adam Scott</empleado>
<plan_celular>premium</plan_celular>
...
<tiempo>129</tiempo>
<valor>235
<empleado>
```

Son los mismos datos, pero el vocabulario difiere. En la empresa telefónica Adam Scott es un cliente, pero en la empresa donde trabaja es un empleado.

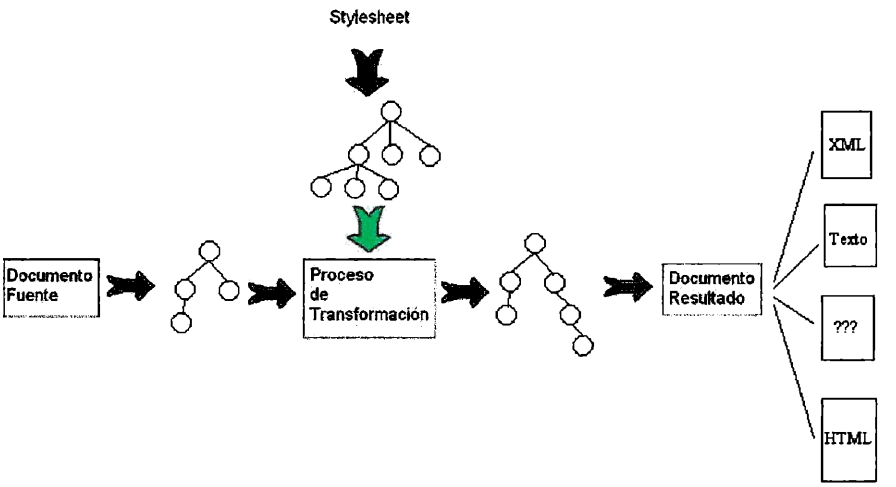
Observación: Puede ocurrir que todas las aplicaciones intercambien sus datos usando XML y que aun así se necesite transformar el XML.

Modelo de procesamiento XSLT

La tarea principal de un procesador XSLT es aplicar una stylesheet a un documento fuente XML y producir un documento resultado. Esto se muestra en el siguiente diagrama.



El documento resultado producido por el procesador puede ser un archivo XML, un archivo de texto, un archivo HTML o un archivo con otro formato.



XSLT define sus operaciones en términos de una representación de un documento XML denominada árbol. La especificación no define la representaciόm de datos, sólo el modelo conceptual que define los objetos en el árbol, sus propiedades y relaciones.

Las reglas de conformancia especifican que un procesador XSLT debe ser capaz de leer una stylesheet y usarla para transformar un árbol fuente en uno resultado. En la práctica los procesadores crean el árbol fuente a partir del documento XML fuente y crean un documento XML resultado a partir del árbol resultado.

Ejemplo

Supongamos que tenemos que crear una página HTML con los datos de las referencias que aparecen en la tesis. En principio contamos con un archivo XML, llamado tesisReferencia.xml, que está bien formado y contiene los datos de todas las referencias usadas. También tenemos el procesador Instant Saxon.

Luego hacemos un programa, la stylesheet, para indicar al procesador cómo debe mostrar los datos.

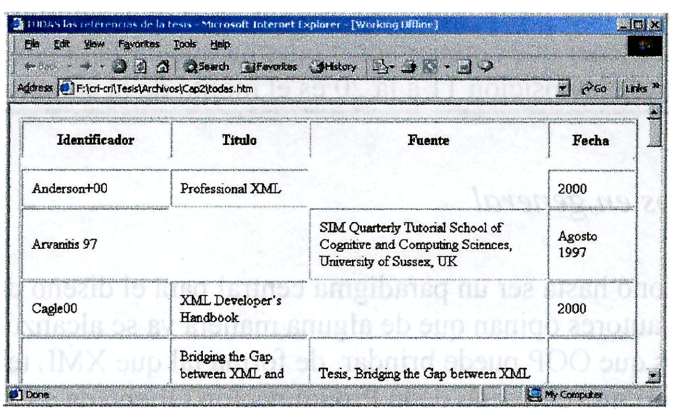
```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet
      version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
      <html>
      <head>
            <title>TODAS las referencias de la tesis</title>
      </head>
      <body>
  <table border="2" cellpadding="10">
   <tr>

<th>Identificador</th><th>Título</th><th>Fuente</th><th>Fecha</th
>
   </tr>
    <tr>
   <xsl:for-each select="//referencia">
      <tr>
          <td><xsl:value-of select="identificador"/</td>
          <td><xsl:value-of select="titulo"/></td>
         <td><xsl:value-of select="fuente"/></td>
         <td><xsl:value-of select="fecha"/></td>
      </tr>
   </xsl:for-each>
      </tr>
   </table>
      </body>
      </html>
</xsl:template>
</xsl:stylesheet>
```

En este momento ejecutamos el procesador indicándole que use los datos del archivo tesisReferencia.xml y las instrucciones de la stylesheet imprimirTodos.xsl para generar una página htm llamada todas.htm. Para esto en la línea de comandos escribimos

```
C:\>saxon tesisReferencia.xml imprimirTodos.xsl > todas.htm
```

Finalmente un browser nos muestra la página *todas.htm*



Identificador	Título	Fuente	Fecha
Anderson+00	Professional XML		2000
Arvanitis 97		SIM Quarterly Tutorial School of Cognitive and Computing Sciences, University of Sussex, UK	Agosto 1997
Cagle00	XML Developer's Handbook		2000
	Bridging the Gap between XML and	Tesis, Bridging the Gap between XML	

La dupla XSLT-XPATH es una herramienta útil para *personalizar sitios Web*. Por ejemplo con ellos podemos seleccionar en el servidor únicamente la información de

un documento XML que el usuario está autorizado a ver (XPATH). Y luego transformar los datos y presentarlos como él desee.

Siempre se va a necesitar hacer cosas como extraer la dirección de una orden de compra y agregarla a una factura. Entonces, la comunicación entre empresas para hacer e-Commerce se transformará cada vez más en un caso de definir cómo extraer y combinar los datos desde un conjunto de documentos XML para generar otro conjunto de documentos: XSLT es la herramienta ideal para ese trabajo. [Kay00]

DOM

Introducción al DOM

El DOM (Document Object Model) provee un conjunto de objetos para representar un documento XML como un árbol así como también provee métodos y propiedades que se pueden usar para manipular los objetos.

El DOM es la clave para manipular el contenido de un documento XML porque es lo que permite que el contenido de los documentos XML estén disponibles desde los programas.

Provee una visión estructurada en forma de árbol del documento. [Anderson00]

DOM es el nombre genérico de una librería que arma un árbol con el contenido de un archivo o un string xml. Para armar dicho árbol, parsea el xml y expone un conjunto de interfases para que el programador maneje los elementos del árbol, es decir para que obtenga datos o manipule los objetos del árbol que el DOM armó con el xml.

Lo importante que hace el DOM es permitir al programador que obtenga datos de ese árbol **ordenados en una jerarquía** que arma sin tener que capturarlos él por programa (es decir sin tener que obtener por programa que hasta la posición 10 es el código de cliente, desde la posición 11 a la 20 es el monto de la compra, etc.)

Modelo de Objetos en general

OOP evolucionó hasta ser un paradigma central para el diseño de sistemas complejos. Algunos autores opinan que de alguna manera ya se alcanzó el umbral del límite de las ventajas que OOP puede brindar, de forma tal que XML tendrá un rol significativo en el giro que el diseño de la Programación Orientada a Objetos está tomando.

Una *clase* define que propiedades, métodos y eventos están disponibles para un objeto. Las propiedades son características de un objeto. Los *métodos* son las acciones que el objeto puede tomar y los *eventos* son situaciones a las cuales el objeto responde, es decir actividades que desencadenan una respuesta del objeto. Por otra parte el *objeto* posee los contenidos reales de sus propiedades. Mientras que la *interfaz* es la descripción de la clase. Especifica qué propiedades poseen los objetos que pertenezcan a la clase, pero no tiene código que especifique el comportamiento. Finalmente un *objeto* es la *instancia* de una clase.

Modelo de Objetos de XML

Con la Programación Orientada a Objetos tenemos un espacio de objetos y para ese espacio se define un modelo de objetos. Así cada objeto puede tener propiedades, métodos (acciones que puede tomar) y eventos (a los que responde). También en este espacio de objetos se definen *relaciones* que los objetos pertenecientes a este espacio tienen entre sí.

El modelo de objetos para XML es el **XML Document Object Model** conocido con el nombre de XML DOM.

XML DOM está conformado por 23 objetos, cada uno expone un interfaz mediante la cual se puede acceder a los objetos en los programas de manera tal de manejar documentos, elementos, atributos, nodos, etc.

Las interfases son independientes del lenguaje de programación.

Ejemplo:

> El XML DOM contiene la interfase DOMDocument. Esta interfase permite crear y manipular documentos XML DOM.
>
> Otra interfaz que ofrece el XML DOM es IXMLDOMParseError. Esta interfaz permite consultar al objeto XML y determinar (si hubo) cuáles fueron los errores que ocurrieron al analizar con un parser el documento.

Las principales estructuras componentes del documento son los nodos en el árbol de objetos. Para acceder a los items y manipularlos se navega un árbol que fue parseado usando las interfases de DOM. Es decir que usamos DOM para manipular el contenido de un documento XML. Se puede encontrar más información respecto a este tema en [Anderson99] [Cagle00] y [Spencer99]. El W3C está encargado de mantener la recomendación del Document Object Model.

> En conclusión se usa XML DOM para acceder a los objetos XML desde nuestros programas.

Transformaciones estructurales de XML usando el DOM

Implementaciones de DOM

W3C emite una especificación de DOM, pero en cuanto a *implementaciones*, podemos decir que hay más de un DOM, dependiendo de la arquitectura para manipular objetos que se use. Las arquitecturas existentes son Microsoft Component Object Model (COM) y Common Object Request Broker Architecture (CORBA), del Object Management Group (OMG).

El World Wide Web Consortium desarrolló su propio DOM y los principales creadores de browsers (Microsoft y Netscape) se comprometieron a soportar este DOM. Microsoft Internet Explorer 5 y Netscape 6 soportan los aspectos de XML de este DOM. [Spencer99]

Por otra parte Microsoft desarrolló su propio DOM, MSXML DOM es el DOM de Microsoft [Cagle00].

SAX

Simple API for XML. Es la otra API principal para trabajar con documentos XML. Tiene sus orígenes en los inicios del desarrollo de XML y fue creada por desarrolladores que necesitaban una manera eficiente de acceder a los documentos XML en aquel tiempo. Para acceder al contenido del documento XML usa una aproximación distinta de la que toma el DOM. SAX no le da a las aplicaciones una vista de árbol de todo el documento. Para documentos XML enormes esto sería muy ineficiente. SAX notifica al programa que lo está usando a medida que parsea el documento. Ofrece eventos, pero queda a cargo del programa que lo usa decidir qué sucede en respuesta a un evento ofrecido por SAX.

Manipulación de objetos mediante el DOM

En particular con el DOM podemos:

1)Crear un Objeto documento XML vacío
2)Dar entrada a XML
   a) Desde un string
   b) Desde una fuente externa
3)Mostrar archivos XML
   a)Propiedad xml
   b)Método save()
4)Navegar un documento usando el DOM

1)Crear un Objeto Documento XML vacío

En la mayoría de los casos lo primero que se hace al trabajar con XML es crear un objeto documento XML. En el Modelo de Objetos XML de Microsoft crear un objeto DOMDocument. Este objeto será el objeto raíz (root) en el modelo de objetos.

En el ejemplo creamos un objeto e ingresaremos XML desde un string con VBScript, Visual Basic y Jscript

Ejemplo:

VBScript	Visual Basic	Jscript
Dim xmlDoc Set xmlDoc= createObject("Microsoft.X MLDOM")	Dim xmlDoc as DOMDocument Set xmlDoc=new DOMDocument	nodeSource = new ActiveXObject ("Microsoft.XMLDOM");

2)Dar entrada a XML

Una vez que creado el objeto documento XML, este objeto existe, pero no tiene datos. Necesitamos entonces *cargar la data* XML desde algún lugar. Podemos ingresar data XML desde un string de texto formateado según XML (con los tags de apertura y cierre adecuadamente anidados), desde una fuente externa como un archivo, una isla de datos u otro objeto DOM. A continuación mostramos cómo cargar los datos desde un string de texto y desde una fuente externa de datos.

2.a) Desde un string

En este caso el XML se pasa un string de texto formateado adecuadamente a un objeto documento XML. Primero se crea un string con la información en forma de datos entre tags respetando la estructura de árbol que pide la especificación de XML y después se usa el método LoadXML para "cargar" el string. Esta es una técnica apropiada para generar dinámicamente documentos XML especialmente cortos.

Ejemplo:

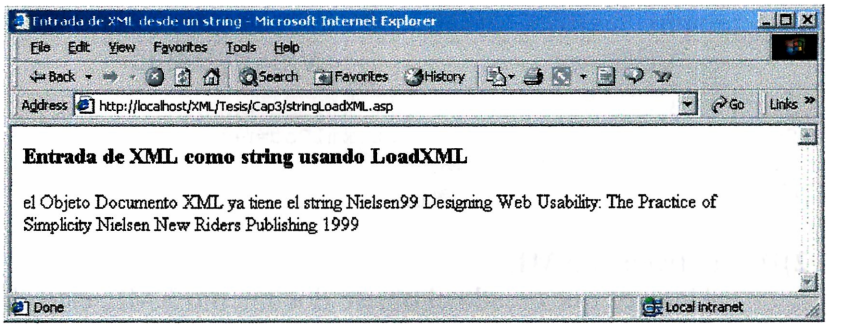
En este ejemplo creamos un objeto documento XML corto usando el método LoadXML del DOM de Microsoft y VBScript.

El archivo con el código se llama *stringLoadXML.asp*.

```
<%@Language="VBScript"%>
<HTML>
<HEAD>
<TITLE>Entrada de XML desde un string </TITLE>
</HEAD>
<BODY>
<h3> Entrada de XML como string usando LoadXML</h3>
<%
Dim xmlDoc
Dim buf
Set xmlDoc=createObject("Microsoft.XMLDOM")
Buf=""
Buf=Buf+"<referencia>"
Buf=Buf+"<identificador>Nielsen99 </identificador>"
Buf=Buf+"<titulo>Designing Web Usability: The Practice of

Simplicity</titulo>"
Buf=Buf+"<autor> Nielsen </autor>"
Buf=Buf+"<editorial>New Riders Publishing </editorial>"
Buf=Buf+"<fecha>1999</fecha>"
Buf=Buf+"</referencia>"
xmlDoc.LoadXML buf
Dim str
str="el Objeto Documento XML ya tiene el string "& xmlDoc.text
Response.Write str
%>
</BODY>
</HTML>
```

Se puede observar que cargamos el string XML en el objeto documento xmlDoc (la raíz) y que en el mismo código se encuentra el string XML que formará la data de xmlDoc.



Al mostrarlo desde el servidor Web, la pantalla que genera *stringLoadXML.asp* aparece en la figura.

2.b) **desde una fuente de datos**

En este caso ingresamos data XML desde una fuente externa a través del método Load. Load permite especificar un nombre de archivo, una URL u otro documento XML como parámetro para cargar con datos el documento.

Ejemplo

De la manera que sigue se cargan de datos de diversas fuentes. Previamente debe haber sido creado el objeto documento XML, `xmlDoc`.

Fuente	Código
Archivo	`xmlDoc.Load "c:\Tesis\tesisReferencia.xml"`
URL	`XmlDoc.Load "\\recurso\Tesis\tesisReferencia.xml"`
Documento XML	`XmlDoc.Load "http://www.ponisio.com/XML/input/ref.xml"`

Ejemplo

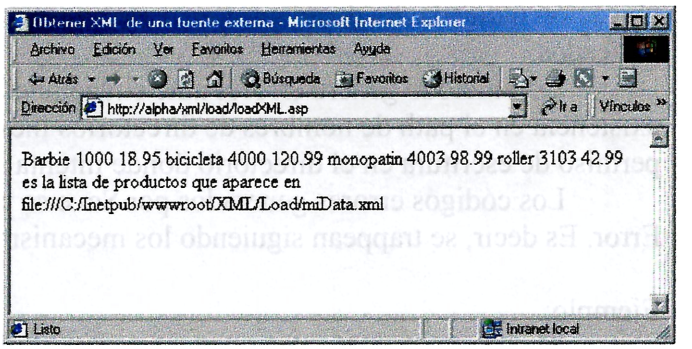
Teniendo un catálogo de productos en un archivo xml, llamamos un archivo asp que se llama *loadXML.asp* desde un browser para mostrar los datos del archivo xml con los productos.

```
<%@Language="VBScript"%>
<HTML>
<HEAD>
<TITLE>Obtener XML de una fuente externa</TITLE>
</HEAD>
<BODY>
<%
Dim xmlDoc
Set xmlDoc=createObject("Microsoft.XMLDOM")
Dim str
' Load de un archivo local:
XmlDoc.Load "C:\Inetpub\wwwroot\XML\Load\miData.xml"

str=""
str= str + XmlDoc.text + " es la lista de productos que aparece en "
str= str + XmlDoc.url
Response.Write str
%>
</BODY>
</HTML>
```

La data está en miData.xml. El contenido de este archivo se encuentra en el Apéndice bajo el título miData.xml, ejemplo de manipulación de objetos mediante el DOM.

El resultado de llamar al archivo se observa en la figura.



3)Mostrar archivos XML

De la misma manera que se puede ingresar data XML desde un string o desde una fuente externa se puede obtener la salida de XML a un string o a una fuente externa.

a)con la propiedad xml

La propiedad xml recorre el árbol XML y hace la representación en formato de string del XML. Si la propiedad PreserveWhiteSpace está en false, xml recorre el árbol inorder y remueve los espacios. Es una propiedad de la clase IXMLDOMNode
Ejemplo:

En este ejemplo mostramos el contenido del objeto documento XML (xmlDOC) mediante la propiedad xml.

```
<%@Language="VBScript"%>
<HTML>
<HEAD>
<TITLE>Salida de XML desde un string </TITLE>
</HEAD>
<BODY>
<h1> Salida de XML como string </h1>
<%
Dim xmlDoc
Dim buf
Set xmlDoc=createObject("Microsoft.XMLDOM")
Buf=""
Buf=Buf+"<referencia>"
Buf=Buf+"<identificador>Nielsen99 </identificador>"
Buf=Buf+"<titulo>Designing Web Usability: The Practice of
Simplicity</titulo>"
Buf=Buf+"<autor> Nielsen </autor>"
Buf=Buf+"<editorial>New Riders Publishing </editorial>"
Buf=Buf+"<fecha>1999</fecha>"
Buf=Buf+"</referencia>"
XmlDoc.LoadXML buf
Response.Write "el
contenido SIN TAGS
de buf es " &
XmlDoc.xml
%>
</BODY>
</HML>
```

La salida que genera se observa en la figura.

b)Método save()

El método save( ) es el recíproco del método load( ) que vimos previamente. Save convierte la representación XML interna en un string y luego salva ese string en la ubicación indicada. Es decir, genera un archivo xml con el nombre y en el directorio indicados.

Este método genera errores ante situaciones como lockeo de permisos o la existencia en el path de nombres de directorios inexistentes. Por ejemplo si no se tiene permiso de escritura en el directorio donde intenta salvar el archivo, genera un error.

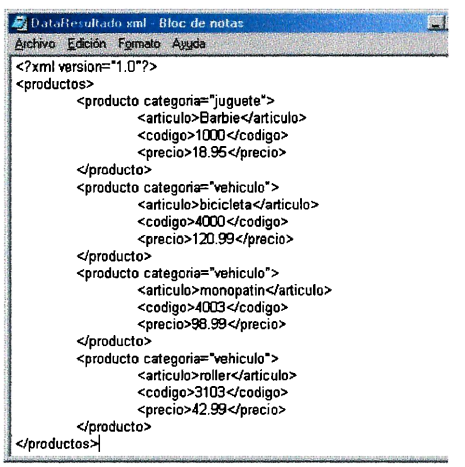
Los códigos errores generados por Save se pueden capturar con la sentencia On Error. Es decir, se trappean siguiendo los mecanismos usuales del lenguaje.

Ejemplo:

Mediante un archivo asp (*salvarXML.asp*) desde un browser creamos un archivo .xml (*DataResultado.xml*) con los datos del archivo xml *miData.xml* que contenía el catálogo de productos.

```
<%@Language="VBScript"%>
<HTML>
<HEAD>
<TITLE>Salvar XML en una fuente externa</TITLE>
</HEAD>
<BODY>
<h1> Salvar XML en DataResultado.xml </h1>
<%
Dim xmlDoc
Set xmlDoc=createObject("Microsoft.XMLDOM")
XmlDoc.Load
"C:\Inetpub\wwwroot\XML\Save\miData.xml"

XmlDoc.save
"C:\Inetpub\wwwroot\XML\Save\DataResultado.xml"
%>
</BODY>
</HTML>
```



DataResultado.xml es el archivo generado gracias al save que aparece en el asp.

4)Navegar un documento usando el DOM

En el DOM, el componente básico es el nodo. Un nodo es un objeto y para recuperar, ingresar, eliminar y modificar datos es necesario navegar el árbol del documento XML manipulando nodos. Para esto el DOM ofrece todo un conjunto de interfaces.

Ejemplo:

En este ejemplo usamos el código que aparece a continuación para navegar el árbol XML del documento miData.xml (el catálogo de juguetes) y generamos una página HTML que muestra el precio de la bicicleta. Como la bicicleta es el segundo producto en la lista de productos del catálogo, buscamos ese nodo en el árbol XML y mostramos el valor que indica el nodo precio del nodo producto correspondiente a la bicicleta.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<TITLE>Navegación del árbol XML generado por el DOM </TITLE>
</HEAD>
<BODY>
<H1>Navegación del árbol XML generado por el DOM</H1>
<%
Dim xmlDoc
Dim nodo1
Dim nodo2
Set xmlDoc=createObject("Microsoft.XMLDOM")
Dim str
' Load desde un archivo local
XmlDoc.Load "C:\Inetpub\wwwroot\XML\Navegar\miData.xml"
'cargo el nodo con el dato del arbol (bicicleta)
Set nodo1=XmlDoc.documentElement.childNodes(1).childNodes(0)
Set nodo2=XmlDoc.documentElement.childNodes(1).childNodes(2)
str="La "
str=str + nodo1.text
str=str + " vale $"
str= str + nodo2.text
Response.Write str
%>
</BODY>
</HTML>
```

El resultado es el que aparece en la figura y el archivo con los productos es miData.xml, que se encuentra en el Apéndice.

Conclusión

Para entender XML hay que dejar de ver el archivo XML como una secuencia y pensar en términos de árboles. Una vez que entendemos que XML es sólo una manera estándar de organizar la información jerárquicamente, resulta natural la actividad que desarrollan los programas que se usan para procesar XML, las especificaciones de los distintos miembros y la conformación de familia XML.

Si bien trabajar con XML requiere un cambio en la manera de pensar la organización de la información, XML presenta varias áreas que prometen revolucionar las técnicas usadas en la actualidad.

XML logró implementar con éxito la vieja idea de usar un estándar para intercambiar información entre plataformas. Como signo positivo, los principales jugadores de la industria de soft lo soportan y alientan su incorporación a los diseños de aplicaciones hipermedia en la Web.

Capítulo 4

Vistas de información

Introducción

La línea entre bases de datos y documentos XML es cada vez más delgada. La transferencia de información entre documentos XML y bases de datos está cada vez más cerca de liberarse de los errores gracias al desarrollo de lenguajes XML de consulta para acceder a documentos XML y a los schemas, que permiten usar tipos en el contenido de los documentos. Por otra parte, XML está diseñado con el objetivo de permitir el intercambio fácil y económico de información entre aplicaciones. En consecuencia nos preguntamos a cerca de los puntos en común entre las bases de datos (y la manera en que ofrecen información) por un lado y la personalización de la información en aplicaciones (principalmente en sitios Web) usando XML por otro. Surge entonces la idea de que en ambos se muestra la información personalizada según sea su destino. Concepto que denominamos *vistas*.

En este capítulo primero explicamos qué es una vista en SQL (Structured Query Language), dado que el análisis en cuestión parte de las ampliamente usadas vistas de las bases de datos relacionales. Después planteamos el problema de programación dinámica y estática en el contexto de XML. Finalmente planteamos una analogía entre SQL y XML, analizamos la relación entre vistas de información y la familia XML, el problema del mantenimiento de la información y el uso de la vistas como herramienta para el intercambio de información entre sistemas heterogéneos.

Definición

Desde el punto de vista de SQL, una vista parece una tabla que conteniendo columnas y se consulta de la misma manera que una tabla. Sin embargo, una vista no tiene datos. Conceptualmente se puede pensar una vista como una máscara cubriendo una o más tablas de manera tal que las columnas en la vista se encuentran en una o más de las tablas subyacentes. Así, las vistas no usan espacio físico para almacenar los datos. La definición de una vista (la cual incluye la consulta en la que se basa, el layout de las columnas y la garantía de privilegios) se almacena en el diccionario de datos.

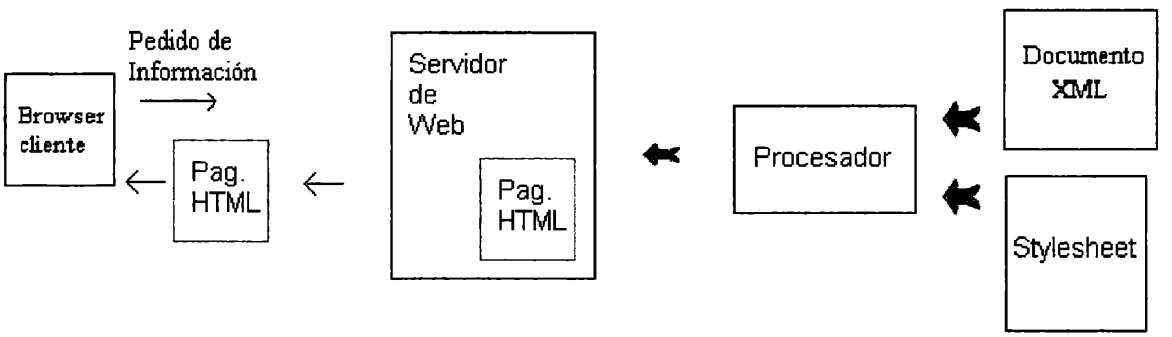
Cuando se consulta una vista, en realidad la vista consulta las tablas en las que se basa y devuelve los valores en el formato y orden especificados en la definición de la

vista. Dado que no hay datos físicos directamente asociadas con ellas, las vistas no se pueden indexar.

Las vistas se usan a menudo para obtener (y garantizar) la seguridad a nivel de fila y de columna sobre los datos. Por ejemplo se puede garantizar a un usuario el acceso a una vista que muestre sólo las filas correspondientes a ese usuario en la tabla, mientras que no se le garantiza al usuario acceso a todas las filas en la tabla. Del mismo modo, mediante la vista se puede limitar las columnas que el usuario puede ver. [Gennick00]

Problema de programación dinámica y estática

Al intentar generar vistas implementadas con XSLT que cumplan con algunas de las funciones de las vistas tan usadas en SQL, se presenta el inconveniente de que la vista como concepto de SQL es dinámica y con XSLT se hacen cosas estáticas. Esto quiere decir que con XSLT partimos de un archivo XML, un archivo XSL (stylesheet), aplicamos la transformación usando el parser y obtenemos el o los archivos XML o HTML. Pero si cambian los datos del archivo XML original (el que usamos como fuente de datos y al que le aplicamos la stylesheet), tenemos que volver a ejecutar el parser para aplicarle la transformación (que se encuentra en la stylesheet, es decir en el archivo XSL). Eso en el mejor de los casos, por supuesto que si modificamos algún tag del archivo original, entonces debemos modificar la stylesheet y después correr el parser con la stylesheet modificada sobre el archivo XML con los datos.



Ahora bien, usando XSLT y un procesador para generar las páginas, podríamos generar la salida cada vez que fuese necesario. Al usar un procesador XSLT, la página HTML que se mostrará puede ser obtenida del procesador + documento XML + stylesheet como resultado de algún evento disparado, por ejemplo que el usuario haya solicitado la información desde una página.

Por supuesto que también podríamos generar una vez la página HTML y almacenarla en el servidor de Web para enviarla como está cuando el usuario lo requiera. La misma situación se da cuando en lugar de mostrar la información mediante un sitio Web, nos encontramos con una aplicación que debe generar resultados. También se puede usar un procesador para que genere una salida de los datos con el formato deseado. Ese procesador se puede disparar una vez y usar su salida cada vez que sea

necesario o llamarlo cada vez que sea necesario para que genere la salida correspondiente a los datos y el formato en el último estado en que se encuentren.

Analogía con SQL

Cuando se usa SQL junto a las bases de datos hay que construir ante todo la base de datos (con sus tablas, restricciones de seguridad, etc.) y los programas (por ejemplo procedimientos) que permiten operar sobre ella, pero también existen herramientas disponibles para ayudar a la administración y mantenimiento de todo el sistema. De la misma forma, al incorporar XML en un sistema hay que construir *etiquetas y reglas* para definir la estructura de los documentos XML bien formados (o válidos, si existe un DTD) así como también hay que construir *programas* para manipular los árboles XML obtenidos (por ejemplo programas que manipulen el árbol XML usando el DOM), pero también hay herramientas gratuitas para trabajar con XML (por ejemplo los procesadores XSLT).

Con respecto a las etiquetas y reglas, las etiquetas pertenecen a vocabularios y junto a las reglas conforman lenguajes XML que es necesario diseñar a la medida del sistema donde se usen. Puede ocurrir que el sistema se encuentre en un contexto específico y común, y que entonces ya se haya definido un lenguaje apropiado. En este caso, en vez de diseñar las etiquetas y reglas desde cero se usan las que ya hayan sido definidas, pero de cualquier manera debe estar presente la decisión de usar ese lenguaje.

Por otra parte al usar XML, igual que al usar SQL para las bases de datos, en muchos casos hay que escribir programas cuya función será manipular el árbol XML presente en cada documento XML.

Sin embargo XML y SQL tienen una diferencia básica: la manera de organizar la información. En este punto SQL recupera información que se encuentra en tablas con filas y columnas mientras que XML, como ya vimos en detalle, tiene la información organizada en forma de árbol. En consecuencia, SQL no almacena naturalmente una jerarquía, hay que programar, incorporar columnas que indiquen jerarquía o darle alguna estructura por programa a la información recuperada.
XML, al contrario almacena naturalmente la información con una estructura jerárquica (el árbol XML), esto hace que en todo momento la información se encuentre organizada jerárquicamente.

SQL permite a los usuarios acceder a información contenida en la base de datos y manipularla de diversas maneras. XML permite consultar documentos XML y árboles XML mediante programas desarrollados usando el DOM y los miembros de la familia XML XPath y XSLT.

Tanto con XML como con SQL se puede:

- Retornar información restringida por filas en SQL y contenido de elementos en XML
- Retornar información restringida por columnas en SQL y por elementos en XML

- Agrupar filas en conjuntos, aplicar una función a cada uno de esos conjuntos de filas y devolver la lista con los resultados. Por ejemplo SELECT sum(total) FROM facturas.
- Ordenar la información recuperada.Por ejemplo SELECT * from REFERENCIAS ORDER BY fuente;

Ejemplo:

Seleccionamos el identificador, titulo, fuente, fecha de las referencias cuya fuente sea `http://www.w3.org`.

Usando SQL:

```
SELECT identificador, titulo, fuente, fecha  FROM referencias WHERE
fuente =' 'http://www.w3.org';
```

Usando XML

```
  <xsl:for-each select="//referencia[fuente='http://www.w3.org']">
      <tr>
          <td><xsl:value-of select="identificador"/></td>
          <td><xsl:value-of select="titulo"/></td>
         <td><xsl:value-of select="fuente"/></td>
         <td><xsl:value-of select="fecha"/></td>
      </tr>
   </xsl:for-each>
```

Ejemplo:

Seleccionamos el título y la fecha de *todas* las referencias.

```
SELECT titulo, fecha  FROM referencias;

   <xsl:for-each select="//referencia">
      <tr>
          <td><xsl:value-of select="titulo"/></td>
         <td><xsl:value-of select="fecha"/></td>
      </tr>
   </xsl:for-each>
```

El análisis de diferencias entre SQL y XML así como también el desarrollo de lenguajes XML es un tema muy amplio que ya fue tenido en cuenta por la W3C. Es así como se hicieron varias propuestas para realizar consultas sobre estructuras XML: una fue XQL, el acrónimo de XML Query Language, sugerido por desarrolladores de AT&T. En esta propuesta se presenta un modelo de recuperación de datos semejante a la notación de SQL (SELECT * FROM tabla), pero no se siguió este camino porque se consideró que este lenguaje no era lo suficientemente flexible.

Otro de los trabajos presentados a la W3C es XML-QL. Se trata de un lenguaje XML que adopta muchas de las técnicas usadas en las bases de datos. Crea estructuras muy parecidas a las de SQL y es muy flexible. Sin embargo tiene dos desventajes

importantes. Una es que no provee una manera explícita de preservar la secuencialidad. Por ejemplo al seleccionar todas las referencias que conforman un arbol cuidadosamente creado y desarrollado para tener las referencias ordenadas por identificador, nada asegura que se obtengan las referencias en ese orden. La otra desventaja es que no preserva la estructura. Esto significa que hay que recrear la estructura del documento resultado en la misma consulta. Es necesario especificar los tags a usar en el documento resultado dentro de la consulta.

Al momento de realizar este trabajo, la W3C está trabajando sobre una especificación denominada XQery. Se trata de un lenguaje de consulta diseñado para ser aplicado en todos los tipos de fuentes de datos XML. Surge la necesidad de un lenguaje con esta característica porque muchas de las propuestas de especificaciones de lenguajes para consulta eran muy apropiadas para los tipos específicos de datos para los cuales fueron diseñadas, pero no cumplían con lo necesario para consultar otros tipos de datos.

Según la especificación de XQuery, "un lenguaje de consulta que use la estructura de XML de manera inteligente debe poder expresar consultas de diversos tipos de datos ya sea almacenados físicamente en (archivos) XML o vistos como XML vía middleware."

Por "diversos tipos de datos" se refiere a fuentes que incluyen documentos estructurados y semi-estructurados, bases de datos relacionales y repositorios de objetos.

Esta especificación tiene estado de "en desarrollo" y se prevee que cuando se encuentre estable será una alternativa a las técnicas actuales y en particular a la usada en este trabajo en el patrón Group By.

Relación entre vistas de información y la familia XML

En cuanto a la relación entre las vistas de información y la familia XML, los miembros de la familia de especificaciones que se usan para generar vistas, es decir, para personalizar la información, son XSLT y XPath. XPath para seleccionar las partes del documento XML necesarias para crear la vista y XSLT para realizar las transformaciones con el objeto de acomodar los datos seleccionados. Por ejemplo si tenemos un documento XML con información de todas las referencias usadas en una tesis y en la misma estructura existe un elemento para cada referencia denominado `<capitulo>` cuyo contenido es el capítulo en el que se llama a la referencia, entonces usando XPath podemos seleccionar únicamente las referencias pertenecientes al capítulo 2.

Una segunda actividad necesaria es transformar los datos dado que a veces el orden de los elementos en el árbol XML no es el mismo orden con el que queremos mostrarlos. Un caso común es cuando almacenamos los elementos en el documento XML por un identificador y queremos mostrarlos ordenados alfabéticamente. Por ejemplo en el tema de las referencias de la tesis, los elementos referencia están ordenados por identificador. Pero como también las referencias incluyen como elementos aquellos que tienen la fuente de la referencia, podríamos generar una vista tal que mostrase todos las referencias agrupadas por fuente.

Motivos para usar las vistas

El motivo principal para usar vistas en un entorno XML es la necesidad de personalizar la información presentada al usuario.
Es una práctica muy común tener la información en un documento XML o en un medio de almacenamiento más importante como por ejemplo una base de datos y generar un documento XML a partir de ésta. Teniendo de una manera u otra el documento XML como depósito de la información, se necesita extraer parte de los datos y presentarlos de una manera específica. Todo según quién sea el usuario.

La solución entonces es generar vistas de la información hechas a la medida del usuario.

Mantenimiento

Mantener información duplicada es una tarea sumamente costosa en tiempo y esfuerzo. Por eso en muchas aplicaciones se opta por tener un solo repositorio de información, por ejemplo una base de datos y generar una o varias vistas de la información adecuada a las necesidades.

Si la información estuviese duplicada, entonces cada modificación que se hiciese en uno de los depósitos debería hacerse también en los demás. Esto implicaría aumentar considerablemente el trabajo de mantenimiento.

Con la información en un solo lugar y múltiples vistas de la información, las actualizaciones o cambios producidos en la fuente real de la información se reflejan en todas las demás consultas a esta fuente.

Necesidad de importación y exportación de la información

Actualmente existe una gran variedad de aplicaciones (especialmente aplicaciones hipermedia en la Web) que requieren importar información ya sea del usuario mediante una página HTML y un browser o de otra aplicación. Un ejemplo de esto es la carga de formularios mediante la Web, desde el ingreso de los datos de un currículum hasta la carga de horas trabajadas por los emplados en una empresa donde los empleados se encuentren dispersos en clientes.

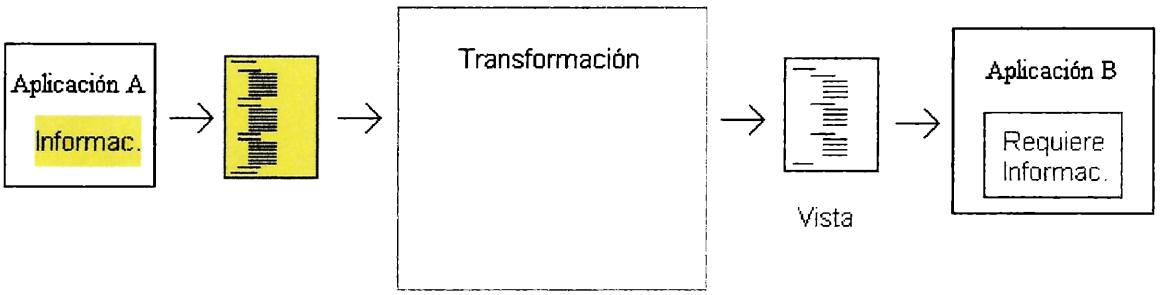
Por otra parte las aplicaciones una vez que recopilaron grandes volúmenes de información, en general deben pasar la información a otra aplicación para que los datos sean procesados o almacenados. Por ejemplo una empresa puede cargar los currículums de los aspirantes y almacenarlos luego en una base de datos para su posterior clasificación y selección. En consecuencia existe una alta y creciente necesidad de realizar importaciones y exportaciones de información.

Hoy en día lo importante es con qué es capaz de comunicarse la aplicación y para eso XML resulta ser una herramienta de alto valor.

Las vistas actúan como adaptadores tanto para la importación como para la exportación dado que son el resultado de transformar y filtrar los datos y su estructura a la medida de las aplicacones que los importan y exportan.

Intercambio de información entre sistemas heterogéneos mediante el uso de vistas

Las vistas posibilitan que dos sistemas que almacenan y procesan la información que no están relacionados entre sí puedan compartir información. Esto se da porque las vistas permiten tomar la información que se necesita comunicar de un sistema y darle el formato que requiere el sistema que la recibe.



De esta manera se crean filtros que transforman la información traduciéndola para que sistemas heterogéneos la puedan intercambiar.

En la Web, los agentes son una manera efectiva de procesar información, por lo tanto se podrían usar para lograr este objetivo.

Capítulo 5

Patrones XML

Presentamos en este capítulo cuatro patrones que ayudan a incorporar XML en el desarrollo de aplicaciones en la Web. Son, en formato de patrones, técnicas para mostrar información personalizada.

El primero, llamado Group by porque imita el trabajo de la cláusula `Group by` que puede aparecer en las sentencias `SELECT` de SQL, describe cómo agrupar la información usando XSLT y XPath.

El segundo se llama Colaborador Externo y explica cómo incorporar comportamiento externo en la presentación de datos XML mediante stylesheets.

El tercero se llama In-Out Tray (Bandeja de Entrada-Salida) porque funciona como un empleado que recibe pedidos en una bandeja de entrada y deposita los resultados en una de salida. Explica la manera de organizar el problema de obtener y mostrar datos independientemente del lenguaje y de la estructura que representa los datos.

El cuarto y último, Orquesta, explota las posibilidades de XML para favorecer el intercambio de información entre aplicaciones.

Cabe aclarar que en general los patrones se pueden agrupar en dos niveles:

- Los de alta granularidad que expresan soluciones sin involucrar los detalles de implementación y
- Los de baja granularidad en donde se tienen en cuenta problemas y sus soluciones con la implementación. Estos son técnicas disponibles en formato de patrón para aprovechar la reusabilidad favorecida por este formato.

Los patrones que aparecen a continuación pertenecen al nivel de patrones de baja granularidad.

Por otra parte, los patrones para vistas se caracterizan por aplicarse en situaciones donde se requiere manejar información que no necesariamente está almacenada con la misma estructura con la que se la desea mostrar. Por ejemplo en el patrón Group By, para mostrar la información hay que generar primero una capa lógica en donde se ordena y obtiene la información.

Group by, patrón de agrupación de la información

Nombre

Group by

Objetivo

Es el típico caso del problema de mostrar items agrupados según alguna característica de los mismos y a partir de una lista.

Una transformación genera un archivo HTML a partir de un archivo XML que contiene los datos y de un archivo XSL especificando qué hacer con los mismos.

Motivación

Se necesita mostrar los elementos agrupados según algún criterio inherente a los mismos mientras que no se los almacena agrupados con el mismo criterio.

A partir de un documento XML que almacena elementos, se desea agrupar los elementos que tienen un subelemento en común y generar una salida con los elementos agrupados. Se usa una stylesheet para generar la transformación deseada.

Un sistema Web en donde los datos se almacenan siguiendo el estándar XML requiere procesamiento de esos datos mediante un parser. La transformación que sufren los datos no varía en el tiempo, pero ante cada pedido los datos sí pueden variar.

Fuerzas

XSLT es un leguaje funcional, no existe la asignación de variables.

Solución

Usar un stylesheet con la transformación que agrupe los elementos.

El archivo XML que tiene los datos, entonces hay que crear otro archivo con la transformación, en este caso el segundo archivo es el stylesheet e indica a un programa específico (parser) qué debe hacer con los datos almacenados en el archivo xml.

La transformación recorre el árbol en donde se encuentran los datos y genera una lista con los grupos que se encuentran en el árbol. No hay grupos repetidos en esa lista.

En un segundo paso la transformación recorre la lista de grupos sin repetir que armó y por cada item de esta lista ingresa una fila en una tabla que se mostrará en la salida generada y recorre a su vez de nuevo todo el árbol del archivo xml con los datos recogiendo y mostrando todos los elementos que pertenecen al grupo actual. De cada elemento imprime solamente una parte en la segunda columna de la fila correspondiente. El grupo al cual pertenece el elemento determina la fila correspondiente.

Como resultado el parser genera un archivo HTML.

Ejemplos

Se dispone de una lista de temas en un archivo XML. Cada tema tiene un formato específico y pertenece a un grupo. Por lo tanto se puede agrupar según el grupo al cual pertenezca.

El formato de los datos en el archivo de datos es:

```
<tema>
        <titulo> Título del tema acá  </titulo>
        <grupo>Grupo al cual pertenece</grupo>
   <descripcion> Descripción del tema</descripcion>
    <solucion>
            ...
    </solucion>
</tema>
```

Un tema posee los subitems `titulo`, `grupo`, `descripcion` y `solucion`. El item que se encuentra más a la derecha pertenece al que está inmediatamente a su izquierda en el nivel de anidamiento.

Los temas pertenecen a una lista:

```
<lista>
      <tema>
            <titulo> Título del tema acá  </titulo>
            <grupo>Grupo al cual pertenece</grupo>
         <descripcion> Descripción del tema</descripcion>
          <solucion>Solución conocida </solucion>
      </tema>
          ...
      <tema>
            <titulo> Título del tema acá  </titulo>
            <grupo>Grupo al cual pertenece</grupo>
         <descripcion> Descripción del tema</descripcion>
          <solucion>Solución conocida </solucion>
      </tema>
</lista>
```

Ejemplo:

```
<lista>
      <tema>
               <titulo>Muestra ordenada de items de una
lista</titulo>
               <grupo>Group By</grupo>
         <descripcion>Este es el clásico problema de agrupamiento
en el cual se
                desea agrupar los items que tienen un tipo de
item en comun.
               </descripcion>
         <solucion>
                  ...
         </solucion>
      </tema>

      <tema>
            <titulo>Selecionar todos los Items del
documento></titulo>
                  <grupo>Selecionar todos los items del
documento</grupo>
            <descripcion>Se necesita mostrar todos los elementos
de un documento
                  XML.
                  </descripcion>
            <solucion>solucion a selecionar todos los Items del
documento
            </solucion>
      </tema>
</lista>
```

Se desea mostrar parte de la información de cada tema, en el ejemplo el título y el grupo al cual pertenece, mediante un browser cualquiera con la particularidad de que los títulos deben estar agrupados por el grupo al cual pertenece el tema.

Archivo XML: *lista.xml*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="lista.xsl"?>
<lista>
      <tema>
            <titulo>Muestra ordenada de items de una lista</titulo>
            <grupo>Group By</grupo>
         <descripcion>Este es el clásico problema de agrupamiento en
el cual se desea agrupar los items que tienen un tipo de item en
comun.</descripcion>
         <solucion>
               ...
         </solucion>
      </tema>
      <tema>
            <titulo>Problema del lado a lado</titulo>
```

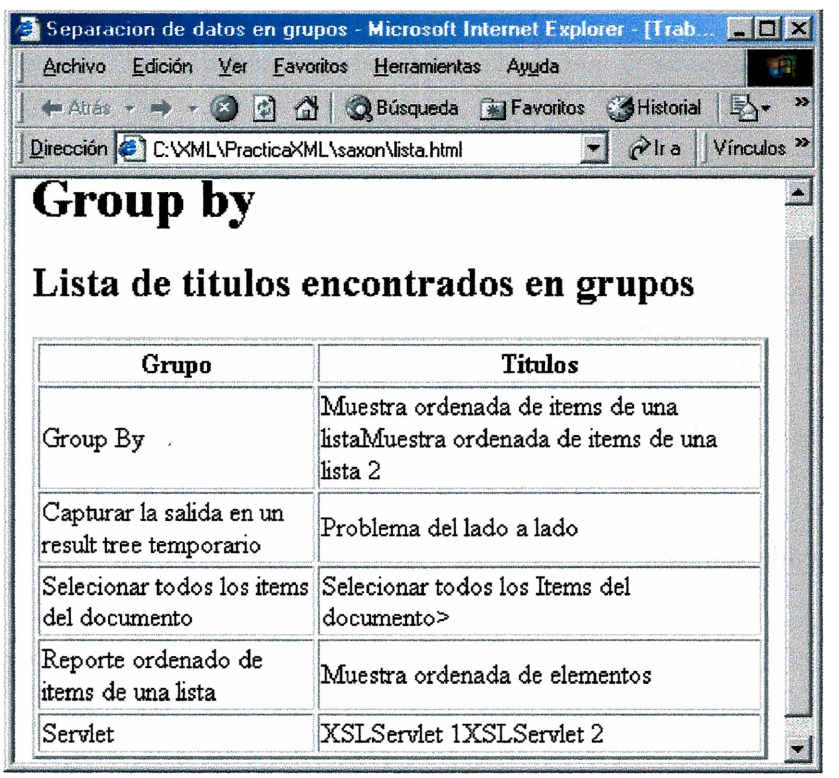
```
            <grupo>Capturar la salida en un result tree
temporario</grupo>
            <descripcion>Se necesita generar salida de texto que
ubique el contenido de los elemento lado a lado.
             </descripcion>
           <solucion>
          ...
                      </solucion>
      </tema>
      <tema>
            <titulo>Muestra ordenada de items de una lista 2</titulo>
            <grupo>Group By</grupo>
          <descripcion> Tengo un ejemplo de XML asi y necesito..
            </descripcion>
            <solucion>blah blah blah </solucion>
        </tema>
      <tema>
          <titulo>Selecionar todos los Items del documento>
                 </titulo>
            <grupo>Selecionar todos los items del documento</grupo>
          <descripcion>Se necesita mostrar todos los elementos de un
documento XML.
            </descripcion>
          <solucion>solucion a selecionar todos los Items del
documento
          </solucion>
      </tema>
      <tema>
            <titulo>Muestra ordenada de elementos</titulo>
            <grupo>Reporte ordenado de items de una lista</grupo>
            <descripcion>Tengo los elementos que quiero mostrar en
una  lista desordenada y necesito mostrarlos ordenados  alfabéticamente
según un campo del elemento</descripcion>
            <solucion>pepepepe</solucion>
      </tema>
      <tema> <titulo>XSLServlet 1</titulo>
             <grupo>Servlet</grupo>
             <descripcion>.......</descripcion>
             <solucion>.......</solucion>
      </tema>
      <tema> <titulo>XSLServlet 2</titulo>
             <grupo>Servlet</grupo>
             <descripcion>.......</descripcion>
             <solucion>.......</solucion>
      </tema>
</lista>
```

Stylesheet aplicada: *lista.xsl*

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
<xsl:template match = "/">
<html>
<head>
   <title>Separacion de datos en grupos</title>
```

```
</head>
<xsl:variable name="grupos-sin-repetir"
      select="/lista/tema[not(grupo=preceding-
sibling::tema/grupo)]/grupo"
/>
<body>
   <h1>Group by</h1>
   <h2>Lista de titulos encontrados en grupos</h2>
   <table border="2">
      <tr>
        <th>Grupo</th><th>Titulos</th>
        </tr>
<xsl:for-each select = "$grupos-sin-repetir">
            <tr>
            <td><xsl:value-of select="."/></td>
            <td><xsl:for-each select = "//tema[grupo=current()]">
                  <xsl:value-of select="titulo"/>
                  </xsl:for-each>
            </td>
        </tr>
      </xsl:for-each>
   </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Página creada: *lista.html*



Consecuencias

Se observa claramente la **separación del contenido respecto de la representación** que la familia XML obliga a hacer, así como también se observan las ventajas que esa separación provee.

Posibilidad de **probar rápidamente un prototipo**. Por ejemplo mientras se prueba la transformación no se necesita conocer la naturaleza de los datos y cuando se modifica la transformación no se realiza ningún cambio en los datos.

Reusabilidad. Además esta transformación se puede aplicar siempre que se produzca la necesidad de agrupamiento mencionada, modificándola ligeramente y sin imponer grandes restricciones en el archivo XML con los datos.

Esto hace que el patrón se ampliamente aplicable a los casos que requieren un "Group by" según la jerga SQL.

Implementación

Sirve para procesar elementos de una lista que se encuentran en interleaving

Usado para ordenar los datos cuando se quiere organizar a los datos con una estructura ligeramente distinta de cómo están almacenados, las variaciones en cuanto a programación son mínimas.

Un ejemplo de la vida real en el que sería útil este patrón es el correo electrónico vía web que permite ordenar por fecha o por autor.

Colaborador Externo, patrón que utiliza comportamiento externo

Es un patrón diseñado para vistas. Explica cómo incorporar comportamiento externo. Aplicable cuando se necesita que al usar la stylesheet se calcule o extraiga información de diversas fuentes, algunas no convencionales (por ejemplo entrada de un usuario o dato de lectora de código de barras).

Nombre

Colaborador Externo

Objetivo

Incorporar objetos al sistema de modo que el comportamiento de estos objetos colabore para generar el resultado final del procesamiento en un sistema que usa XML.

Como muestra se usa una clase Java para incorporar información al resultado de procesar un documento XML.

Es decir que el objetivo es realizar el procesamiento de datos de tal manera que existan objetos que incorporen comportamiento externo a un sistema dinámico de objetos, donde parte de los datos se almacenan siguiendo el estándar XML.

Motivación

Comúnmente se requiere calcular un dato en función de los valores que se recuperan de alguna fuente de datos. El cálculo en sí es una función de los datos que se tienen cuando se está procesando el documento XML y no se puede incorporar información al dtd porque alteraría inútilmente la estructura del mismo contaminando el diseño y alterando la legibilidad.

Es decir que en el momento de procesar el documento XML, se necesita obtener información basada en los datos que aparecen en ese documento.

Fuerzas

El sistema tiene objetos y debe ser flexible.

La flexibilidad del sistema exige que no sólo se acomode o filtre datos almacenados según el estándar XML, sino que genere más información en función de los datos a partir del comportamiento de objetos.

Solución

Invocar en la stylesheet métodos de una clase Java.

Para esto existen varias alternativas, dependiendo de la información que haya que calcular. Las alternativas son:

- invocar métodos estáticos usando como parámetros los datos del documento XML o
- crear una instancia de la clase usando el constructor por defecto y llamar a un método sobre esa instancia de la clase también usando como parámetros los datos obtenidos del documento XML de ser necesario.

Ejemplo

Se usa una función externa escrita en Java.

Se muestra cómo llamar a un método de una clase Java creada por el usuario.

La stylesheet invoca un método de la clase clasejava. El método retorna el número 1 y éste aparece en la página html creada por el parser. El archivo XML usado está vacío y se llama dummy.xml.

Archivo XML: *dummy.xml*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="externa.xsl"?>
<!--Dummy XML para mostrar como se usan las funciones de extensiones de
Java" -->
<dummy>
</dummy>
```

Stylesheet aplicada: *externa.xsl*

```
<xsl:stylesheet
        version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:j="http://www.lauraponisio.com/xml/java/clasejava">
<xsl:template match="/">
   <html>
   <body>
```

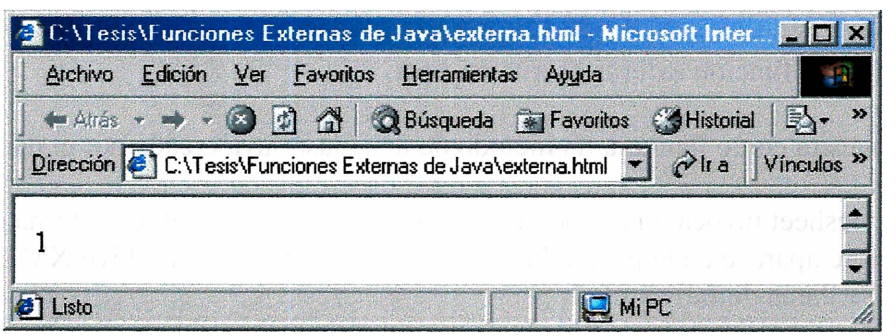
```
      <xsl:variable name="resultado" select="j:metodo()"/>
      <p><xsl:copy-of select="string($resultado)"/></p>
   </body>
   </html>
</xsl:template>

</xsl:stylesheet>
```

Clase invocada: *clasejava*,
fuente:

```
public class
clasejava
{  public static int
metodo()
   {  int n = 1;
      return n;
   }
}
```

Página creada: *externa.html*



Se muestra cómo llamar a un método que ya existe en la librería de clases estándar de Java.

Usando el mismo dummy.xml del ejemplo anterior, la stylesheet usa la clase java.util.Date, crea un objeto Date y lo inicializa con la fecha actual.

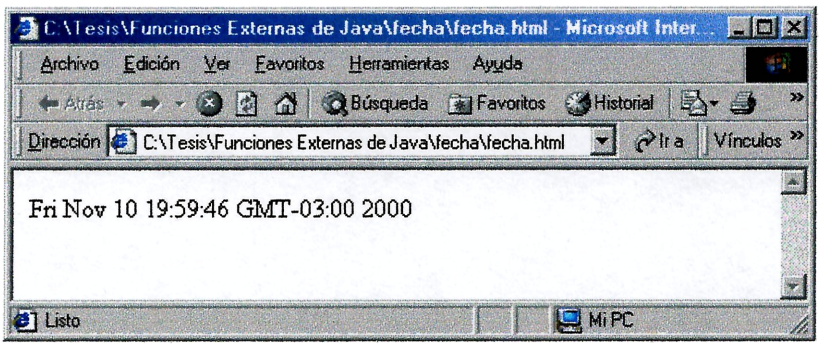
Stylesheet aplicada: *fecha.xsl*

```
<xsl:stylesheet
        version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:fecha="http://www.lauraponisio.com/xml/java/java.util.Date">
<xsl:template match="/">
   <html>
   <body>
```

```
      <p><xsl:value-of select="fecha:to-string(fecha:new())"/></p>
   </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

Página creada: *fecha.html*



Consecuencias

Incorporamos comportamiento externo manteniendo la abstracción de ese comportamiento. Evitamos la mezcla de código necesario para generar la vista con el código externo incorporado.

Mantenemos la modularidad.

Disminuimos la complejidad de la stylesheet sin perder comportamiento. Muchos parsers hoy en día brindan la posibilidad de llamar a métodos de clases Java. La utilidad de este modelo se observa claramente al diseñar sistemas con objetos en donde se necesita que la stylesheet tenga una función mucho más programática (computacional) que simplemente el hecho de mostrar el árbol de datos como aparece en el archivo XML.

Implementación

Este patrón es útil en casos donde la extracción de datos de archivos XML requiera mucho procesamiento. Es decir que la información obtenida de los datos no alcance a lo que se quiere mostrar o guardar y sea necesario realizar cálculos para obtener nuevos datos en función de los datos estáticos.

Si bien el mecanismo aquí usado no está definido en las especificaciones de XPATH ni de XSLT, muy probablemente se estandarice en el futuro dado que figura en la lista de incorporaciones deseadas para la versión 2.0.

En algunos casos se puede implementar junto a patrones usados para ordenar los datos cuando se quiere organizar a los datos con una estructura muy distinta de cómo están almacenados.

Por ejemplo este patrón no se relaciona con el patrón Group by porque éste usa sólo el procesamiento XSLT para agrupar los datos, pero sí se relaciona con patrones en donde el procesamiento con XSLT no alcance y la solución incorpore el llamado a funciones externas.

In Out Tray

Nombre

In-Out Tray

Objetivo

Organiza la actividad entre los componentes que participan en el proceso de obtención y muestra de datos. Plantea el uso de XML como intermediario eficaz entre componentes de la aplicación posibilitando así la independencia del lenguaje y de la estructura que representa los datos.

Motivación

Supongamos que tenemos la tarea de obtener datos de una fuente según un criterio dado y mostrarlos.

Necesitamos recoger el criterio de obtención de datos de alguna entrada (por ejemplo un browser), seleccionar los datos adecuados de una fuente y mostrarlos.

Queremos que el diseño sea lo suficientemente flexible como para que dos componentes computacionales ajenas entre sí intercambien datos. Es decir que exista una puerta en cuanto a los datos de manera tal de obtenerlos de una fuente que puede no tener nada que ver con la aplicación que estamos diseñando y por lo tanto almacenarlos con una estructura extraña para nuestro sistema.

Fuerzas

Queremos maximizar la reusabilidad.

Necesitamos que el diseño sea lo suficientemente abierto como para permitir usar datos que generó otro sistema.

Queremos que el diseño sea modular para que sea flexible y permita manipular los componentes enteros.

También queremos que sea modular para abstraer los componentes y que la modificación de uno sea transparente a los otros.

Solución

Usamos tres componentes y un archivo XML que tiene los datos.

Un componente (In) recibe los datos de entrada, es decir, el criterio de búsqueda.

Un archivo XML se usa como repositorio de los datos o fuente. Puede haber sido generado por cualquier aplicación, inclusive un componente add-on externo que se utilice para facilitar el intercambio de datos vía XML. La única condición es que la estructura del archivo XML debe ser conocida por el segundo componente.

Un segundo componente (worker) recibe el criterio de búsqueda de la componente In y extrae los datos que necesita del archivo XML. Procesa los datos, por ejemplo aplicando una regla de negocio (por ejemplo aplicando un descuento X a los ítems que compró el cliente Y, porque tiene un certificado de descuento). Finalmente este worker envía los resultados obtenidos a un tercer componente (Out).

Out recibe los datos y genera la salida de los mismos.

Out se crea en función del dispositivo que se usará para la salida, igual que In para la entrada.

Además pueden existir simultáneamente varios In y varios Out para recibir los criterios de diversas entradas y enviarlos a distintos dispositivos de salida (browsers, celulares, etc.).

La solución naive es tener una componente que reciba, busque los datos en la fuente, los procese y genere la salida. En este caso se producen muchas entradas en la fuente de datos buscando la información lo cual dificulta la optimización y deteriora la performance. El diseño se hace menos modular con un elevado grado de acoplamiento lo que disminuye la abstracción y por lo tanto disminuye también la flexibilidad y la reusabilidad. En este caso un cambio por pequeño que sea compromete todo el sistema.

Por otra parte en la solución naive no se usa XML desaprovechando así la posibilidad de intercambio prolijo de datos con aplicaciones extrañas y obliga a tener un mayor conocimiento de la estructura de los datos en la fuente que los provee, así como también exige mayor programación y acoplamiento.

Ejemplo

Muestra de productos de un catálogo por Web.

Tenemos una lista de productos en un archivo XML.

El formato de los datos en el archivo de datos es:

```
<fila>
      <columna>libro</columna>
      <columna>2103</columna>
      <columna>42.99</columna>
</fila>
```

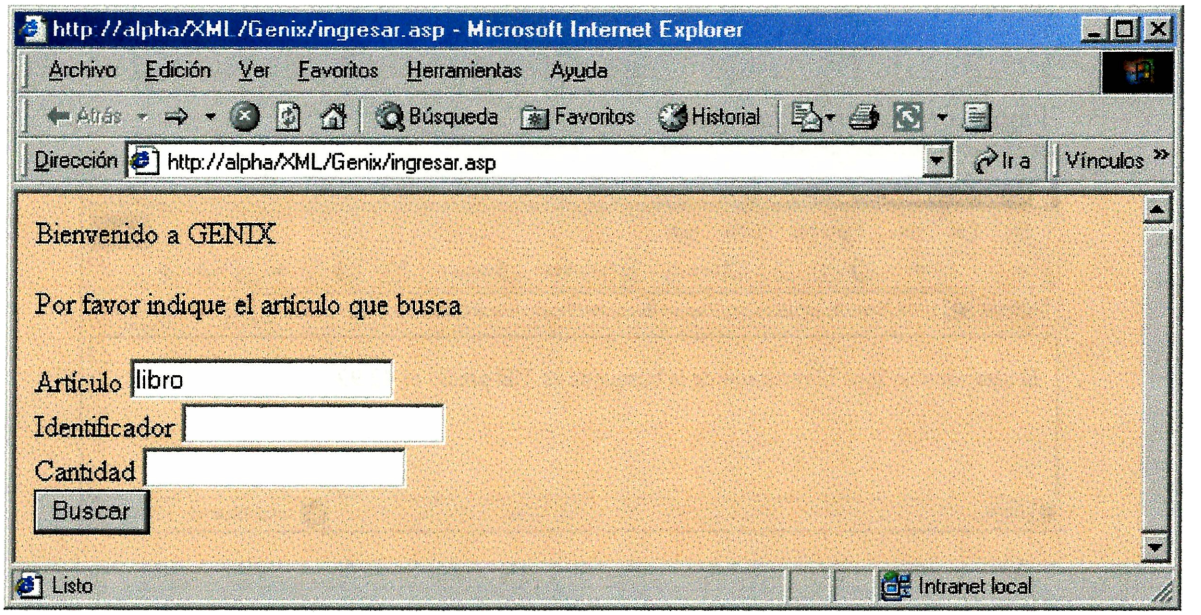
Deseamos mostrar parte de los datos que están almacenados en *genData.xml*.

Para esto, recibimos el criterio de búsqueda en un form de un archivo asp (*ingresar.asp*) que cumple el papel de componente In.

Mediante el archivo *traerdatos.asp* buscamos la información en el archivo XML y la mostramos.

Se unieron el segundo y tercer componente en uno merced a lo simple que es el ejemplo.

Página para ingresar criterio: *ingresar.html*

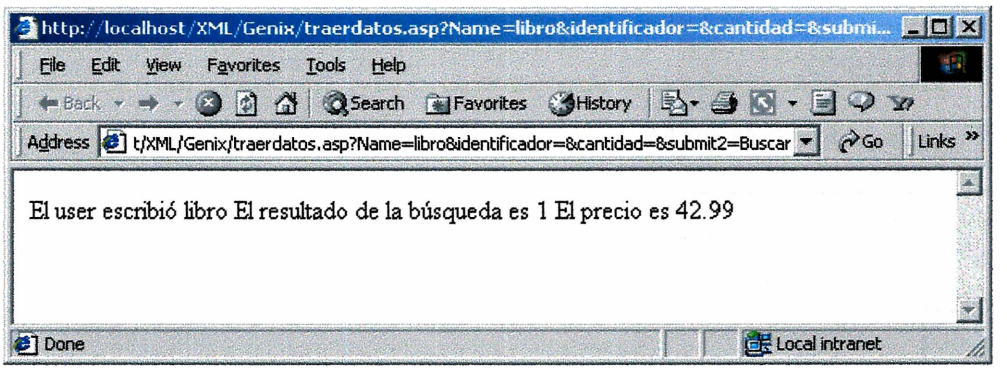


Archivo XML: *genData.xml*

```
<?xml version="1.0"?>
<resultados>
      <fila>
            <columna>Barbie</columna>
            <columna>1000</columna>
            <columna>18.95</columna>
      </fila>
      <fila>
            <columna>auto</columna>
            <columna>111</columna>
            <columna>12000</columna>
      </fila>
      <fila>
            <columna>bicicleta</columna>
            <columna>4000</columna>
            <columna>120.99</columna>
      </fila>
      <fila>
            <columna>monopatin</columna>
            <columna>4003</columna>
            <columna>98.99</columna>
      </fila>
      <fila>
            <columna>roller</columna>
            <columna>3103</columna>
            <columna>42.99</columna>
```

```
        </fila>
        <fila>
                <columna>libro</columna>
                <columna>2103</columna>
                <columna>42.99</columna>
        </fila>
</resultados>
```

Página creada: *traerdatos.asp*



Archivo In (primer componente): *ingresar.asp*

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
</HEAD>
<BODY bgcolor=Peachpuff>
<P>Bienvenido a GENIX </P>
<P>Por favor indique el artículo que busca </P>

<FORM ACTION="traerdatos.asp" METHOD="get" id=form1 name=form1>
Artículo <INPUT name="Name" >
<br>
Identificador <INPUT name="identificador" >
<br>
Cantidad <INPUT name="cantidad" > <BR>

<INPUT type="submit" value="Buscar" id=submit2 name=submit2>
</FORM>

</BODY>
</HTML>
```

Archivo Out (Segundo y Tercer componente): *traerdatos.asp*

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
```

```
</HEAD>
<BODY>

<%
'Busco en el arbol XML
strName =  Request.Form ("Name")
For intLoop = 1 To Request.Form.Count
Next
strName = Request.QueryString("Name")



'Busco los valores de las pistas que recupere arriba
Dim xmlDoc
Dim nodeList
Set xmlDoc=createObject("Microsoft.XMLDOM")
Dim str
' Load desde un archivo local
xmlDoc.Load "C:\Inetpub\wwwroot\XML\GENIX\genData.xml"
'cargo el nodo con el dato del arbol SEGUN lo que el usuario puso en el
form
Set nodeList=
xmlDoc.selectNodes("resultados/fila[columna='"+strName+"']/columna[2]")
str=nodeList.item(0).text
%>
El user escribió <% =strName %>
El resultado de la búsqueda es <% =nodeList.length %>
El precio es <%=str %>
<P> </P>

</BODY>
</HTML>
```

Consecuencias

Nos abstraemos de la representación de datos en la fuente.

Organizamos la solución en componentes abstractos.

Obtenemos máxima performance gracias a organizar los componentes según sea más conveniente.

Generamos un diseño reusable porque obtenemos un diseño flexible que permite acomodar sus componentes según sea necesario.

Obtenemos una solución modularizada y por lo tanto fácil de actualizar y optimizar.

Implementación

No hay características específicas del lenguaje. Por lo tanto este patrón no obliga a ser implementado con un lenguaje u otro.

El uso de XML favorece la independencia del lenguaje aún de sistemas extraños que actúen como fuente de datos. XML hace el papel de "puerta" entre el sistema que implemente este diseño y los sistemas vecinos con los que tenga que interactuar.

Los componentes que participan en este patrón se pueden combinar entre sí cuando la situación lo justifique, por ejemplo por sencillez de la implementación.

Observación

Para describir el patrón se tomó como modelo Design Patterns, elements of Reusable Object-Oriented Software de Gamma, Helm, Johnson y Vlissides

Orquesta, patrón para compartir información.

Nombre

Orquesta

Objetivo

Permitir la obtención de información de distintas fuentes.

Sugiere que los sitios web creen como parte de su interfase una conexión para que el usuario obtenga una copia de los datos una vez que accedió a la información y ordenó el traspaso de datos.

Plantea el uso de XML como herramienta para compartir información, es decir, como un intermediario eficaz para integrar la información entre aplicaciones que no tienen nada en común.

El objetivo también es dar un paso adelante hacia la integración y colaboración de aplicaciones y usuarios mediante la Web.

Motivación

La tarea de recopilar datos de distintas fuentes en Internet hoy se hace manualmente, es decir que una vez que encuentra la información buscada en un sitio, el usuario debe copiarla (a un documento, en un anotador, etc.), imprimirla o memorizarla. Debe luego repetir este procedimiento para completar la información que busca, si ésta se encuentra distribuida entre varios sitios.

Esta labor es tediosa y propensa a errores.

Se podría decir que Internet funciona como un mainframe porque se ve una página por vez y es difícil capturar datos personalizados. Para unir datos el usuario tiene prácticamente que garabatearlos en un anotador.

Fuerzas

Necesitamos que aplicaciones ajenas entre sí compartan información.

Necesitamos rapidez y eficiencia en la transferencia de dartos, es decir buena performance, dado que un sitio puede ser consultado simultáneamente por muchos clientes y no hay que hacer esperar a ninguno.

Queremos que se transporte la información de varias aplicaciones a una en un formato entendible por todas, pero sin perder el formato en la operación. Queremos que la información se traduzca a y se brinde en un formato "universal" y queremos usar esos datos con formato (que generó otro sistema).

Necesitamos una cierta validación intrínseca de los datos transportados. Para esto queremos que el formato de la información transportada permita detectar si se obtuvieron todos los datos pedidos o no. Por ejemplo si se transporta un resumen de operaciones, el formato de transporte de los datos en sí debe indicar si se transfirieron todas las operaciones o si se cortó la transmisión antes de finalizar.

Queremos que el diseño sea modular para abstraer las funciones de ofrecer información (en los sitios que la ofrecen) y recopilarla (en el sitio cliente del usuario), de las demás funciones de los respectivos sitios.

Necesitamos que la solución sea fácil de implementar desde el punto de vista de los sitios que ofrecen y recogen información, porque ingresar el módulo de conexión de datos necesario implica más programación, y el nuevo módulo no debe interferir con el diseño del sitio. Por ejemplo no debe interferir con la capa de navegación. A este nivel, sólo se incorpora la posibilidad de acceder al componente que ofrece la información.

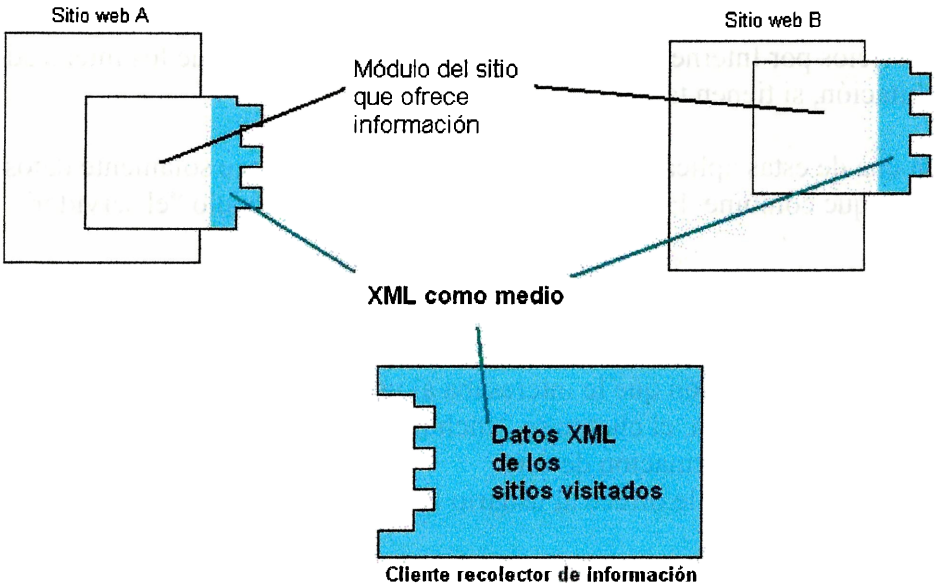
Solución

Que cada aplicación que desee ofrecer información incorpore un módulo del cual se pueda tomar la información con formato. Este módulo interactúa con un cliente guiado por el usuario. Ambos se acoplan y permiten el traspaso de la información usando XML.

Cada sitio que brinde información debe proveer un módulo que interactúe con el cliente. Es decir que los sitios que ofrecen información deben tener un objeto especial que la ofrezca de manera tal que el cliente recolector de información interactúe con él.

Se crea entonces una aplicación cliente que consulta a aplicaciones servidoras. El cliente se conecta en principio a un servidor por vez.

La aplicación cliente que guía el usuario se conecta con las aplicaciones que brindan la información deseada mediante un módulo que se "enchufa" al módulo que ofrecen las aplicaciones consultadas.



Los dos tipos de módulos combinan perfectamente entre sí y son capaces uno de dar y otro de tomar información en forma de datos con formato XML.

En una implementación mejorada el cliente y el servidor pueden negociar el formato de los datos, informando el servidor al cliente un formato específico. Otra mejora en la implementación sería que el cliente pudiera conectarse a varios servidores al mismo tiempo.

La solución naíf es no hacer nada y obligar al usuario a copiar la información por sus propios medios. Finalmente en la solución naíf el usuario tiene que acceder a cada página de su interés de a una página por vez, copiar la información deseada (en general es texto) y pegarla en algún archivo que previamente haya creado con el fin de guardar la información. O simplemente anotarla en un papel.

Ejemplo

Recuperación de datos de dos aplicaciones de la Web y almacenamiento de la información obtenida en un archivo XML.

Tenemos un programa cliente que responde a un usuario que está planeando sus vacaciones. La secretaría de turismo provee a los visitantes de su sitio de un módulo que el usuario lanza cuando encuentra la información acerca de traslados y estadía que le interesa. En adelante llamaremos a este módulo "el cliente". El usuario navega libremente en la Web y cuando llega a una página con los datos que necesita, lanza el cliente.

En la Web, mientras tanto, hay infinidad de sitios que ofrecen servicios al turista. En particular AcmeBus ofrece sus pasajes y un número de teléfono para realizar la venta

mediante algunas tarjetas de crédito. Por otra parte el Hotel Jamaica de Uruguay también ofrece sus servicios por Internet, y da un número de teléfono para que los interesados reserven habitación, si tienen tarjeta de crédito.

Cada una de estas aplicaciones tiene un módulo que ofrece solamente datos a un módulo cliente que combine. En adelante llamaremos a este módulo "el servidor".

Cuando se pone en acción el cliente, se activa el servidor del sitio indicado y se traspasa la información con formato de XML desde el servidor al cliente. Primero el usuario activa el cliente para obtener datos de AcmeBus y luego de la operación el cliente ya posee la información que le interesaba al usuario. Después el usuario lanza el cliente hacia el hotel Jamaica, el cliente se "enchufa" al servidor del sitio del hotel Jamaica y así obtiene la información de éste.

Finalmente el cliente le ofrece al usuario la información recolectada.

El componente XML del módulo del cliente es un destino a donde van a parar los datos obtenidos. Tenemos de esta manera la información recopilada en un archivo XML.

El formato del depósito de datos en el archivo XML es:

```
<?xml version = "1.0"?>
<!DOCTYPE info [
<!ELEMENT info (sitio)*>
<!ELEMENT sitio (nombre?, datos, fecha, URL)>
<!ELEMENT datos (dato+)>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT dato (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT nombre (#PCDATA)>
]>
```

El formato de la información se podría perfeccionar si la aplicación lo mereciera. Por ejemplo los componentes que conforman la interfase podrían negociar un formato específico y detallado de datos mediante un DTD o un Schema. De esta manera, la aplicación que brinda la información informaría a la que consulta el formato de datos y los datos se transferirían con el formato exacto.

Ejemplo

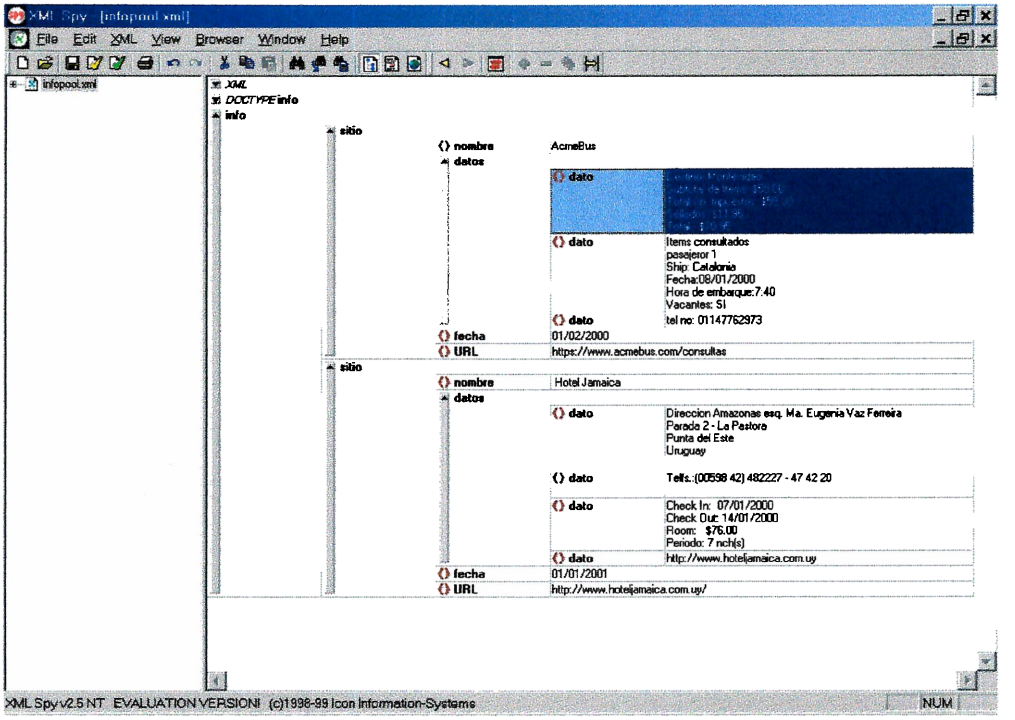
Siguiendo con el ejemplo anterior, el archivo XML que hace de depósito de datos quedaría como el archivo *infopool.xml*, así:

```
<?xml version="1.0"?>
<!DOCTYPE info [
      <!ELEMENT info (sitio)*>
      <!ELEMENT sitio (nombre?, datos, fecha, URL)>
      <!ELEMENT datos (dato+)>
```

```
        <!ELEMENT URL (#PCDATA)>
        <!ELEMENT dato (#PCDATA)>
        <!ELEMENT fecha (#PCDATA)>
        <!ELEMENT nombre (#PCDATA)>
]>
<info>
        <sitio>
                <nombre>AcmeBus</nombre>
                <datos>
                        <dato>Destino: Montevideo
                                Subtotal de Items: $59.00
                                Total sin Impuestos: $59.00
                                Sellados: $11.95
                                Total : $70.95
                        </dato>
                        <dato>Items consultados
                                pasajero 1
                                Ship: Catalonia
                                Fecha:08/01/2000
                                Hora de embarque:7:40
                                Vacantes: SI
                                </dato>
                        <dato>tel no: 01147762973</dato>
                </datos>
                <fecha>01/02/2000</fecha>
                <URL>https://www.acmebus.com/consultas</URL>
        </sitio>
        <sitio>
                <nombre> Hotel Jamaica</nombre>
                <datos>
                        <dato>Direccion Amazonas esq. Ma. Eugenia Vaz Ferreira
                                Parada 2 - La Pastora
                                Punta del Este
                                Uruguay
                        </dato>
                        <dato>Telfs.:(00598 42) 482227 - 47 42 20</dato>
                        <dato>Check In:  07/01/2000
                                Check Out: 14/01/2000
                                Room:   $76.00
                                Periodo: 7 nch(s)</dato>
                        <dato>http://www.hoteljamaica.com.uy</dato>
                </datos>
                <fecha>01/01/2001</fecha>
                <URL>http://www.hoteljamaica.com.uy/ </URL>
        </sitio>
</info>
```

Nota: el texto del archivo infopool.xml copiado arriba tiene espacios en blanco que no tiene el infopool.xml original. Esto es para que se lea e interprete mejor.

El mismo archivo *infopool.xml* visto con un editor de xml (xmlspy, en este caso) se observa así:



El cliente (es decir el módulo que recolecta información de distintos sitios) puede dejar los datos XML obtenidos en un archivo en formato de xml (un "punto XML"). Así el usuario podría editarlo y verlo con un editor de texto, con un editor de XML o con una aplicación creada específicamente para procesar la información obtenida por "el cliente".

Además de recuperar la información y de que se pueda usar en infinidad de aplicaciones, el hecho de dejarla en un archivo XML posibilita que se adose nueva información durante otra sesión, con lo cual el usuario seguiría teniendo la información centralizada y a su alcance.

Obs.: Este ejemplo fue creado especialmente para este patrón. Es sencillo (sólo figura una aplicación cliente y dos aplicaciones consultadas), para explicar las ventajas de usar este patrón, pero las mismas ventajas se multiplican si el contexto es más complicado, por ejemplo si son varios los sitios de la Web de donde se requiere información y son varios los clientes que la reclaman simultáneamente.

Ejemplo con compras

Supongamos que el usuario no sólo quiere obtener información de los sitios, sino que con algunos también realiza transacciones y quiere obtener un registro de la transacción para sí. En este caso hizo dos compras por Internet, compró un libro en Amazon y registró un dominio. Las dos operaciones requirieron un pago mediante la tarjeta de crédito.

Una vez que se realizaron las operaciones, el cliente actuaría de la misma manera que en ejemplo anterior. El archivo XML, destino de los datos recopilados, quedaría como pool.xml, cuyo contenido aparece debajo.

Archivo XML: *pool.xml*

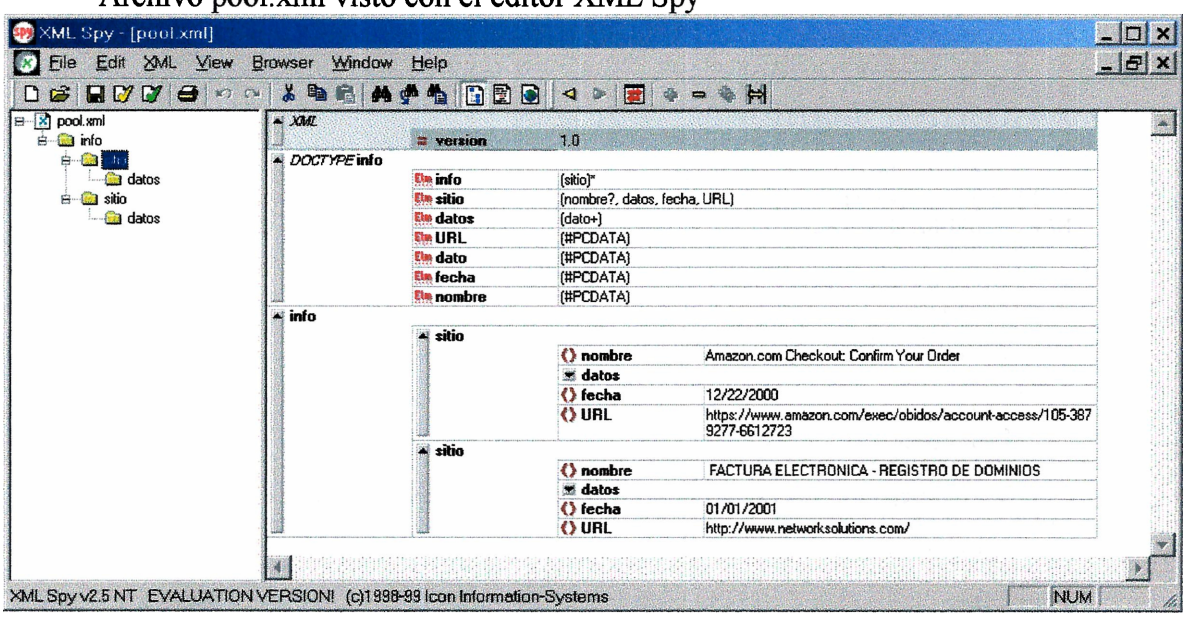
```
<?xml version = "1.0"?>
<!DOCTYPE info [
<!ELEMENT info (sitio)*>
<!ELEMENT sitio (nombre?, datos, fecha, URL)>
<!ELEMENT datos (dato+)>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT dato (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT nombre (#PCDATA)>
]>
<info>
  <sitio>
    <nombre>Amazon.com Checkout: Confirm Your Order</nombre>
    <datos>
      <dato>order #  105-3879277-6612723</dato>
      <dato>Purchase Details
Order #1: Maria Laura Ponisio
Maria Laura Ponisio
Peru 566 2°F
Capital Federal Buenos Aires C1142ACH
Argentina
Subtotal of Items: $16.00
Shipping &amp;amp; Handling: $12.95
Total Before Tax: $28.95
Tax: $0.00
Total for this Address: $28.95
Shipping method: WorldMail
Shipping Preference: Ship this when complete
      </dato>
      <dato>Order Items
Unfinished Tales
J. R. R. Tolkien, Christopher Tolkien(Editor)
$16.00
Availability: Usually ships in 24 hours
      </dato>
      <dato>Purchase Summary
Billing Address:
MARIA LAURA PONISIO
22 N° 270
La Plata Buenos Aires 1900
Argentina
    To be paid by Credit Card
Visa: ****-****-***05876
EXP: 11/2001
      </dato>
    </datos>
    <fecha>12/22/2000</fecha>
    <URL>https://www.amazon.com/exec/obidos/account-access/105-3879277-
6612723</URL>
  </sitio>

  <sitio>
    <nombre> FACTURA ELECTRONICA - REGISTRO DE DOMINIOS</nombre>

    <datos>
```

```
      <dato>Atencion a: Nombre: Maria Laura Ponisio</dato>
      <dato> Account Holder:
Maria Laura Ponisio
22 N 270
La Plata, Buenos Aires
1900
AR
      </dato>
      <dato>De:
Network Solutions, Inc.
505 Huntmar Park Drive
Herndon, Virginia 20170
USA
Customer Service
International
+1 703 742-0914
      </dato>
      <dato>Account Number: 3252665
Password: unlimon
Vendor Password: ujc6695
Purchase Date:  12/07/2000
Dominio: ponisio.com
1 Matching E-Mail Account(s)
Total:   $ 60.00
Periodo: 1 Anyo(s)
      </dato>
      <dato>NameServer1:  ns1.adnbox.com.ar
NameServer2:  ns2.adnbox.com.ar
      </dato>
      <dato>http://www.networksolutions.com/makechanges</dato>
    </datos>
    <fecha>01/01/2001</fecha>
    <URL>http://www.networksolutions.com/ </URL>
</sitio>
</info>
```

Archivo pool.xml visto con el editor XML Spy



Consecuencias

Obtenemos una solución **modular** que permite compartir datos con bajo acoplamiento.

Un cliente, más aplicaciones. La aplicación cliente puede obtener y almacenar datos de distintas aplicaciones-fuente sin modificar el código. Con ligeros cambios podría hacer lo mismo de fuentes no convencionales.

Una aplicación, más clientes. La aplicación que ofrece datos amplía su espectro de clientes. Ahora con cambios ínfimos puede ofrecer información a diversos dispositivos (por ejemplo PDA,Personal Digital Assistant; celulares; etc.). Estos dispositivos que consulten también deberán tener el módulo que haga de interfase con la aplicación que ofrece la información.

No interferimos en el **diseño de navegación** (sólo ingresamos la posibilidad de usar un módulo).

Obtenemos **flexibilidad** en el formato de **transmisión** y almacenamiento de datos.

Reutilizamos datos respetando el formato.

Nos abstraemos de la representación de datos en la fuente.

Reducimos la interacción entre las aplicaciones.

Implementación

Para compartir información se usa el formato XML. Es el medio de traspaso de información entre los sitios que ofrecen información y el cliente que la recolecta.

XML es el medio ideal para implementar el traspaso de la información estructurada. Con XML se facilita la orquestación de datos online y offline. Por otra parte es también la tecnología que facilita la integración y colaboración de computadoras.

Tanto el módulo cliente como el módulo de un servidor tienen que tener una parte para compartir y otra con datos que no se comparten (por ejemplo información acerca de quién accedió).

El uso de XML favorece la independencia del lenguaje aún de sistemas extraños que actúen como fuente de datos. Además como se observa en el ejemplo, el formato XML se puede usar como depósito de la información recolectada y desde allí leer la información con un editor de texto, uno de xml u otra aplicación que acepte xml. También se podría hacer una aplicación específica para procesar datos en formato XML, según se necesite.

Conclusión

XML es una familia de especificaciones que brinda un formato estándar para el intercambio universal de datos. Existen varias maneras de usar XML, cada una tiene sus pro y sus contra. Hay que saber analizar cada caso particular para aplicar el mejor esquema de solución. Las situaciones y los esquemas de solución son muy variados, ergo las maneras de aplicar XML en soluciones también. La amplitud del tema podría llevar a usar XML de una manera intrincada y errónea. Hemos observado que se pueden documentar soluciones XML con patrones de una manera natural y simple por lo tanto no estaría muy lejos la posibilidad de contar con guías en donde se cataloguen soluciones para aplicar XML. Es de suponer que esto agilizaría la implementación de soluciones XML al mismo tiempo que aumentaría la confianza en esta tecnología.

Ante todo, una de las claves para entender XML es que tenemos que acostumbrarnos a ver el documento XML con la forma de un árbol.

A partir de este punto, el paso siguiente es aceptar que a XML es necesario agregarle comportamiento en forma de programas que manipulen los nodos del árbol XML que aparece en el documento XML de la misma manera que es necesario escribir programas para que la base de datos funcione como queremos. Tenemos que escribir esos programas o eventualmente conseguirlos de terceras partes. En este punto los beneficios de la dupla XML-patrones son evidentes, dado que estos programas con sus contextos responden a soluciones clásicas que podrían ser documentadas mediante patrones.

Otro paso a dar para captar la idea de XML y los beneficios que puede aportar usado correctamente es acostumbrarnos a pensar en términos de data organizada jerárquicamente en contraposición a la conocida organización tabular de la data en las bases de datos.

Como se observa en este trabajo, podemos y debemos explotar XML como herramienta para manejar data más formato.

De esta manera podemos aprovechar la capacidad que nos brinda de almacenar datos una vez y presentarlos de distintas maneras (ya sea según los privilegios el usuario que la reciba o el tipo de pantalla donde se muestre, por ejemplo) sobre todo para planear la *personalización* de sitios Web.

Otra capacidad de XML que podemos aprovechar, como se demuestra en este trabajo, es la de almacenar y transportar data con formato para automatizar el intercambio de información entre aplicaciones y de esta manera permitir que aplicaciones trabajando juntas se potencien unas a otras.

No podemos olvidar a la hora de listar las conclusiones obtenidas a partir de este trabajo la capacidad de XML de permitir generar un lenguaje común en el dominio de la aplicación. Esto se transforma en la práctica en un medio que favorece la comunicación y el intercambio de ideas entre desarrolladores. Este beneficio aumenta si consideramos

que según la metodología actual los desarrolladores pueden estar separados en tiempo y espacio, con lo cual un buen lenguaje común de comunicación puede ser crítico.

Todo lleva a concluir que XML puede ser una herramienta estratégica en la nueva generación de sitios Web. Pero para esto es necesario documentar la experiencia de los pioneros en XML, de manera tal que cada vez más desarrolladores continúen el camino con facilidad. Los patrones demostraron ser útiles en este aspecto. Las soluciones en donde se usa XML de manera correcta pueden ser tan sorprendentemente sencillas como engorrosas las soluciones que no usan el aspecto de XML que corresponde al problema.

Si a todo esto sumamos el advenimiento al uso común de dispositivos poco usuales hasta hace un tiempo, como celulares y palm pilots, pero que también necesitan intercambiar información en forma de datos más formato de manera eficiente y que necesitan recibir los mismos datos que otros dispositivos, pero adaptados a su manera de mostrarla, entonces XML resulta aún más atractivo. La idea de que la data viaje a través de aplicaciones sin sufrir los límes de la plataforma hace que el concepto de estandarización propuesto por XML parezca ser una obligación para desarrollos importantes de sistemas integrados en Internet.

Perspectiva a futuro

XML dejará de ser un nombre de moda para convertirse en una parte obligatoria de desarrollos en la Web. La próxima generación de sitios Web combinará las diversas maneras de operar con XML (por no decir tecnologías) presionada por la creciente necesidad de intercambiar información.

Al mismo tiempo que el comercio electrónico madure, los desarrollos en la Web deberán encontrar maneras cada vez más eficientes de integrar la presencia en la Web con los sistemas de negocios. Esa es la razón de usar un formato estándar para compartir universalmente los datos: XML. A medida que XML disminuya la velocidad con que se generan y modifican sus especificaciones y se transforme en una tecnología estable, pasará a ser reconocido y utilizado como una potente herramienta. Pero no se utilizará por estar de moda, sino en donde corresponda. Para saber cómo y dónde conviene usar XML, serán de utilidad los patrones XML para el desarrollo de aplicaciones en la Web.

Referencias

[Anderson+00] Anderson, R., Birbeck, M., Kay, M., Livingstone, S., Loesgen, B., Martin, D., Mohr, S., Ozu, N., Peat, B., Pinnock, J., Stark, P., Williams, K., *Professional XML*, Wrox, 2000.

[Arvanitis 97] Arvanitis, T., N., , SIM Quarterly Tutorial School of Cognitive and Computing Sciences, University of Sussex, UK , Agosto 1997.

[Awai00] Awai, M., Bortniker, M., Carnell, J., Dillon, S., Erwin, D., Goodman, J., Hólm, B., Horton, A., Hubeny, F., Kyte, T., Mitchel II, G., Mukhar, K., Nicol, G., O'Connor, D., Ruth-Hammond, G., Zucca, M., *Professional XML Oracle 8i Application Programming*, Wrox, 2000.

[Cagle00] Cagle, K., *XML Developer's Handbook*, SYBEX, 2000.

[De Muynck00] De Muynck, W., *Bridging the Gap between XML and Hypermedia: a Layered Transformational Approach*, Tesis, Bridging the Gap between XML and Hypermedia: a Layered Transformational Approach, Vrije Universiteit Brussel, Belgium, 2000.

[Gennick+00] Gennick, J., McCullough-Dieter, C., Linker, G., *Oracle8i DBA Bible*, IDG Books Worldwide, Inc, 2000.

[Gamma+95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns*, Addison-Wesley, 1995.

[Goldfarb+98] Goldfarb, A., Prescod, P., *The XML Handbook*, Prentice-Hall Inc., 1998.

[Homer99] Anderson, R., Blexrud, C., Chiarelli, A., Denault, D., Homer, A., Esposito, D., Francis, B., Gibbs, M., Kropog, B., McQueen, C., Reilly, G., Robinson, S., Schenken, J., Sonderegger, D., *Professional Active Server Pages 3.0*, Wrox, 1999.

[Kay00] Kay, M., *XSLT Programmer's Reference*, Wrox, 2000.

[Maruyama+99] Maruyama, H., Tamura, K., Uramoto, N., *Developing Web Applications*, Addison-Wesley, 1999.

[Microsoft] *The Microsoft XML Web site*, http://msdn.Microsoft.com/xml, 08/2000.

[Myers] Myers, T., Nakhimovsky, A., *Professional Java XML Programming with Servlets and JSP*, Wrox, 1999.

[Nielsen99] Nielsen, J., *Designing Web Usability: The Practice of Simplicity*, New Riders Publishing, 1999.

[Oasis] Robin, C., *The XML Cover Pages*, http://www.oasis-open.org/cover/xml.html, 01/12/2000.

[OTN] *Oracle xml parser*, http://technet.oracle.com/, 01/3/2000.

[Sunderraman] Sunderraman, R., *Oracle programming: a primer*, Addison-Wesley, 10/1998.

[Rossi+98] Rossi, G., Schwabe, D., *An Object Oriented Approach to Web-Based Application Design*, Theory and Practice on Object Systems. Wiley and Sons, October 1998.

[Rossi+99] Rossi, G., Lyardet, F., Schwabe, D., *"Patterns for designing navigable Information Spaces" Pattern Languages of Programs IV*, Addison-Wesley, 1999.
[Rossi+00] Rossi, G., Lyardet, F., Schwabe, D., *Patterns for e-commerce applications*, EuroPLoP'2000, 2000.
[Saxon] Kay, M., *About Saxon*, http://users.iclway.co.uk/mhkay/saxon/instant.html, 1999.
[Schwabe99] Schwabe, ., D Rossi, G., Lyardet, F., *Improving Web information systems with navigational patterns*, Computer Networks 31 pp.1667- 1678, 1999.
[Spencer99] Spencer, P., *Professional XML Design and Implementation*, Wrox, 1999.
[SUN] *W3C XML Technical Recommendation Quick Reference Sorted by Name* , http://www.sun.com/xml/standards/trqr/, 25/10/2000.
[TECHWEB] Gonsalves, A., *Vendedores importantes lanzan iniciativa B-to-B*, http://www.techweb.com/news, 25/10/2000.
[W3CDOM] *Document Object Model (DOM) Level 2 Specification Version 1.0 Candidate Recommendation of the World Wide Web Consortium*, http://www.w3.org/TR/REC-DOM-Level-1, 01/10/98.
[W3CInfoset] http://www.w3.org/TR/xml-infoset, .
[W3CNamespaces] *Extensible Markup Language (XML) Specification Version 1.0 Recommendation of the World Wide Web Consortium*, http://www.w3.org/TR/REC-xml-names, 14/01/1999.
[W3CXLink] *XML Linking Language (XLink) Version 1.0 World Wide Web Consortium*, http://www.w3.org/TR/xlink, 03/07/2000.
[W3CXML] *Extensible Markup Language (XML) Specification Version 1.0 Recommendation of the World Wide Web Consortium*, http://www.w3.org/TR/REC-xml, 10/02/1998.
[W3CXPATH99] *XML Path Language (XPath) Version 1.0 Recommendation of the World Wide Web Consortium*, http://www.w3.org, 16/11/1999.
[W3CXPointer] *XML Pointer Language (XPointer) Version 1.0 World Wide Web Consortium*, http://www.w3.org/TR/xptr, 07/06/2000.
[W3CXSL] *Extensible Stylesheet Language (XSL) Version 1.0 W3C Candidate Recommendation* , http://www.w3.org/TR/xsl/, 21/11/2000.
[W3CXSLT] *XSL Transformations (XSLT) Version 1.0 Recommendation of the World Wide Web Consortium*, http://www.w3.org, 16/11/1999.
[W3CXMLSCHEMA1] *XML Schemas Working Draft of the World Wide Web Consortium*, http://www.w3c.org/TR/xmlschema-1, 5/11/1999.
[W3CXMLSCHEMA2] *XML Schemas Working Draft of the World Wide Web Consortium*, http://www.w3.org/TR/xmlschema-2 , 5/11/1999.
[XMLMAGAZINE] *XML Magazine Summer 2000 Vol. 1 Nro. 3*, http://www.xmlmagazine.com, 07/2000.
[XMLPAT] *XML Patterns*, http://www.xmlpatterns.com, 2000.

Apéndice

Implementación de los ejemplos

Ejemplo de documento XML

Archivo *agendaBien.xml*, del ejemplo perteneciente al Capítulo 2, XML, punto "Los documentos describen su propia sintaxis".

```
<?xml version="1.0"?>
<contactos>
   <contacto>
         <usuario>Maria Laura</usuario>
         <direcciones>
               <direccion>
                     <apellido>Gonzalez</apellido>
                     <nombre>Maria Eugenia</nombre>
                           <particular>00000112</particular>
                           <mail>mjk@tutopia.com</mail>
               </direccion>
               <direccion>
                     <apellido>Morales</apellido>
                     <nombre>Claudina</nombre>
                     <particular>00000122</particular>
                     <mail>clau@tutopia.com</mail>
               </direccion>
               <direccion>
                     <apellido>Perez</apellido>
                     <nombre>Carla</nombre>
                     <particular>00000132</particular>
                     <mail>carla@elcharco.com</mail>
               </direccion>
         </direcciones>
   </contacto>
</contactos>
```

miData.xml, ejemplo de manipulación de objetos mediante el DOM

```
<?xml version="1.0"?>
<productos>
      <producto categoria="juguete">
            <articulo>Barbie</articulo>
            <codigo>1000</codigo>
            <precio>18.95</precio>
      </producto>
```

```
        <producto categoria="vehiculo">
                <articulo>bicicleta</articulo>
                <codigo>4000</codigo>
                <precio>120.99</precio>
        </producto>
        <producto categoria="vehiculo">
                <articulo>monopatin</articulo>
                <codigo>4003</codigo>
                <precio>98.99</precio>
        </producto>
        <producto categoria="vehiculo">
                <articulo>roller</articulo>
                <codigo>3103</codigo>
                <precio>42.99</precio>
        </producto>
</productos>
```

Referencias de la Tesis como aparecen en los ejemplos

A continuación se encuentra parte del texto del archivo *tesisReferencia.xml*. Este archivo es el que figura en varios ejemplos del capítulo de la familia XML y del capítulo de Vistas. Como contiene las referencias que aparecen en la tesis, fue creciendo a medida que avanzaba el Trabajo de Grado y se consultaba más bibliografía, en consecuencia los resultados mostrados en las imágenes de los ejemplos pueden variar lijeramente. Aún sigue creciendo así que hay que tomarlo como un material útil para los ejemplos, no como la bibliografía en sí. La bilbliografía definitiva se encuentra en la sección Referencias. Así mismo en algunos ejemplos la estructura varía apenas. Nos tomamos esta licencia para que el ejemplo fuera más gráfico y fácil de entender. Además recortamos parte del archivo para que no fuera tan largo inútilmente. El objetivo es que se observe un ejemplo real de un documento XML.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- creado por Maria Laura Ponisio
http://www.lauraponisio.com.ar -->
<referencias>
  <referencia>
    <!--libro-->
    <identificador>Anderson+00</identificador>
    <autores>
      <autor>
        <apellidos>
          <apellido>Anderson</apellido>
        </apellidos>
        <iniciales>
          <inicial>R</inicial>
        </iniciales>
      </autor>
      <autor>
        <apellidos>
          <apellido>Birbeck</apellido>
        </apellidos>
        <iniciales>
          <inicial>M</inicial>
        </iniciales>
      </autor>
      <autor>
        <apellidos>
          <apellido>Kay</apellido>
        </apellidos>
        <iniciales>
          <inicial>M</inicial>
        </iniciales>
      </autor>
      <autor>
        <apellidos>
          <apellido>Livingstone</apellido>
        </apellidos>
        <iniciales>
          <inicial>S</inicial>
        </iniciales>
      </autor>
    </autores>
    <titulo>Professional XML</titulo>
    <editorial>Wrox</editorial>
    <fecha>2000</fecha>
  </referencia>
  <referencia>
    <!--libro-->
    <identificador>Cagle00</identificador>
    <autores>
      <autor>
        <apellidos>
          <apellido>Cagle</apellido>
        </apellidos>
        <iniciales>
```

```
        <inicial>K</inicial>
      </iniciales>
    </autor>
  </autores>
  <titulo>XML Developer´s Handbook</titulo>
  <editorial>SYBEX</editorial>
  <fecha>2000</fecha>
</referencia>
<referencia>
  <!--libro-->
  <identificador>Gennick+00</identificador>
  <autores>
    <autor>
      <apellidos>
        <apellido>Gennick</apellido>
      </apellidos>
      <iniciales>
        <inicial>J</inicial>
      </iniciales>
    </autor>
    <autor>
      <apellidos>
        <apellido>McCullough-Dieter</apellido>
      </apellidos>
      <iniciales>
        <inicial>C</inicial>
      </iniciales>
    </autor>
    <autor>
      <apellidos>
        <apellido>Linker</apellido>
      </apellidos>
      <iniciales>
        <inicial>G</inicial>
      </iniciales>
    </autor>
  </autores>
  <titulo>Oracle8i DBA Bible</titulo>
  <editorial>IDG Books Worldwide,
Inc</editorial>
  <fecha>2000</fecha>
</referencia>
<referencia>
  <!--libro-->
  <identificador>Gamma+95</identificador>
  <autores>
    <autor>
      <apellidos>
        <apellido>Gamma</apellido>
      </apellidos>
      <iniciales>
        <inicial>E</inicial>
      </iniciales>
    </autor>
    <autor>
      <apellidos>
        <apellido>Helm</apellido>
      </apellidos>
      <iniciales>
        <inicial>R</inicial>
      </iniciales>
    </autor>
    <autor>
      <apellidos>
        <apellido>Johnson</apellido>
      </apellidos>
      <iniciales>
        <inicial>R</inicial>
      </iniciales>
    </autor>
    <autor>
      <apellidos>
        <apellido>Vlissides</apellido>
      </apellidos>
      <iniciales>
        <inicial>J</inicial>
      </iniciales>
    </autor>
  </autores>
  <titulo>Design Patterns</titulo>
  <editorial>Addison-Wesley</editorial>
  <fecha>1995</fecha>
</referencia>
<referencia>
  <!--libro-->
  <identificador>Goldfarb+98</identificador>
  <autores>
    <autor>
      <apellidos>
        <apellido>Goldfarb</apellido>
      </apellidos>
      <iniciales>
        <inicial>A</inicial>
      </iniciales>
    </autor>
    <autor>
      <apellidos>
        <apellido>Prescod</apellido>
      </apellidos>
      <iniciales>
        <inicial>P</inicial>
      </iniciales>
    </autor>
  </autores>
  <titulo>The XML Handbook</titulo>
  <editorial>Prentice-Hall Inc.</editorial>
  <fecha>1998</fecha>
</referencia>
<referencia>
  <!--libro-->
  <identificador>Nielsen99</identificador>
  <autores>
    <autor>
      <apellidos>
        <apellido>Nielsen</apellido>
      </apellidos>
      <iniciales>
        <inicial>J</inicial>
      </iniciales>
    </autor>
  </autores>
  <titulo>Designing Web Usability: The Practice
of Simplicity</titulo>
  <editorial>New Riders Publishing</editorial>
  <fecha>1999</fecha>
</referencia>
<referencia>
  <!—articulo-->
  <identificador>Rossi+00</identificador>
  <autores>
    <autor>
      <apellidos>
        <apellido>Rossi</apellido>
      </apellidos>
      <iniciales>
        <inicial>G</inicial>
      </iniciales>
    </autor>
    <autor>
      <apellidos>
        <apellido>Lyardet</apellido>
      </apellidos>
      <iniciales>
        <inicial>F</inicial>
      </iniciales>
    </autor>
    <autor>
      <apellidos>
        <apellido>Schwabe</apellido>
      </apellidos>
      <iniciales>
        <inicial>D</inicial>
      </iniciales>
    </autor>
  </autores>
  <titulo>Patterns for e-commerce
applications</titulo>
  <fuente>EuroPLoP'2000</fuente>
  <fecha>2000</fecha>
</referencia>
<referencia>
  <!--articulo-->
```

```
    <identificador>Schwabe99</identificador>
    <autores>
      <autor>
        <apellidos>
          <apellido>Schwabe</apellido>
        </apellidos>
        <iniciales>
          <inicial/>D
      </iniciales>
      </autor>
      <autor>
        <apellidos>
          <apellido>Rossi</apellido>
        </apellidos>
        <iniciales>
          <inicial>G</inicial>
        </iniciales>
      </autor>
      <autor>
        <apellidos>
          <apellido>Lyardet</apellido>
        </apellidos>
        <iniciales>
          <inicial>F</inicial>
        </iniciales>
      </autor>
    </autores>
    <titulo>Improving Web information systems with
navigational patterns</titulo>
    <fuente>Computer Networks 31 pp.1667-
1678</fuente>
    <fecha>1999</fecha>
  </referencia>
  <referencia>
    <!--libro-->
    <identificador>Spencer99</identificador>
    <autores>
      <autor>
        <apellidos>
          <apellido>Spencer</apellido>
        </apellidos>
        <iniciales>
          <inicial>P</inicial>
        </iniciales>
      </autor>
    </autores>
    <titulo>Professional XML Design and
Implementation</titulo>
    <editorial>Wrox</editorial>
    <fecha>1999</fecha>
  </referencia>
  <referencia>
    <!--articulo-->
    <identificador>W3CNamespaces</identificador>
    <autores>
      <autor>
        <apellidos>
          <apellido/>
        </apellidos>
        <iniciales>
          <inicial/>
        </iniciales>
      </autor>
    </autores>
    <titulo>Extensible Markup Language (XML)
Specification Version 1.0 Recommendation of the
World Wide Web Consortium</titulo>
    <fuente>http://www.w3.org/TR/REC-xml-
names</fuente>
    <fecha> 14/01/1999</fecha>
  </referencia>
  <referencia>
    <!--articulo-->
    <identificador>W3CXLink</identificador>
    <autores>
      <autor>
        <apellidos>
          <apellido/>
        </apellidos>
        <iniciales>
          <inicial/>
        </iniciales>
      </autor>
    </autores>
    <titulo>XML Linking Language (XLink) Version
1.0
 World Wide Web Consortium</titulo>
    <fuente>http://www.w3.org/TR/xlink</fuente>
    <fecha>03/07/2000</fecha>
  </referencia>
  <referencia>
    <!--articulo-->
    <identificador>W3CXML</identificador>
    <autores>
      <autor>
        <apellidos>
          <apellido/>
        </apellidos>
        <iniciales>
          <inicial/>
        </iniciales>
      </autor>
    </autores>
    <titulo>Extensible Markup Language (XML)
Specification Version 1.0 Recommendation of the
World Wide Web Consortium</titulo>
    <fuente>http://www.w3.org/TR/REC-xml</fuente>
    <fecha>10/02/1998</fecha>
  </referencia>
  <referencia>
    <!--articulo-->
    <identificador>XMLPAT</identificador>
    <autores>
      <autor>
        <apellidos>
          <apellido/>
        </apellidos>
        <iniciales>
          <inicial/>
        </iniciales>
      </autor>
    </autores>
    <titulo>XML Patterns</titulo>
    <fuente>http://www.xmlpatterns.com</fuente>
    <fecha>2000</fecha>
  </referencia>
</referencias>
```

Stylesheet que se usó para obtener las referencias

En la sección Referencias podemos observar las referencias de la tesis. El código de estas referencias es el de *tesisReferencia.xml,* pero el código que le da formato a los datos se encuentra a continuación en el archivo *print_Ref_final.xsl.*

```
<?xml version="1.0" encoding="iso-
8859-1"?>
```

```
<xsl:stylesheet
      version="1.0"
```

```
      xmlns:xsl="http://www.w3.org/
1999/XSL/Transform">

<xsl:template match="/">
      <html>
      <head>
            <title>Page title goes
here</title>
      </head>
      <body>

   <xsl:for-each
select="//referencia">


<xsl:apply-templates/>

   </xsl:for-each>

      </body>
      </html>
</xsl:template>

<xsl:template
match="identificador">
<B>
<xsl:text>[</xsl:text>
<xsl:value-of select="."/>
<xsl:text>]</xsl:text>
</B>
</xsl:template>

<xsl:template match="apellido">
<xsl:value-of select="."/>
<xsl:text>,</xsl:text>
</xsl:template>

<xsl:template match="inicial">
<xsl:value-of select="."/>
<xsl:text>.</xsl:text>
<xsl:text>,</xsl:text>
<xsl:text> </xsl:text>
</xsl:template>

<xsl:template match="titulo">
<i>
<xsl:value-of select="."/></i>
<xsl:text>,</xsl:text>
<xsl:text>  </xsl:text>
</xsl:template>

<xsl:template match="editorial">
<xsl:value-of select="."/>
<xsl:text>,</xsl:text>
<xsl:text>  </xsl:text>
</xsl:template>

<xsl:template match="fuente">
<xsl:value-of select="."/>
<xsl:text>,</xsl:text>
<xsl:text> </xsl:text>
</xsl:template>

<xsl:template match="fecha">
<xsl:value-of select="."/>
<xsl:text>.</xsl:text>
<DIV> &#x0D;</DIV>
</xsl:template>

</xsl:stylesheet>
```

Ejemplo de Group By

Capítulo 5, nombre del patrón: Group By,
Nombre del archivo: *lista.html*

```
<html>
   <head>
      <title>Separacion de datos en grupos</title>
   </head>
   <body>
      <h1>Group by</h1>
      <h2>Lista de titulos encontrados en grupos</h2>
      <table border="2">
         <tr>
            <th>Grupo</th>
            <th>Titulos</th>
         </tr>
         <tr>
            <td>Group By</td>
            <td>Muestra ordenada de items de una listaMuestra ordenada de
items de una lista 2</td>
         </tr>
         <tr>
```

```
            <td>Capturar la salida en un result tree temporario</td>
            <td>Problema del lado a lado</td>
         </tr>
         <tr>
            <td>Selecionar todos los items del documento</td>
            <td>Selecionar todos los Items del documento></td>
         </tr>
         <tr>
            <td>Reporte ordenado de items de una lista</td>
            <td>Muestra ordenada de elementos</td>
         </tr>
         <tr>
            <td>Servlet</td>
            <td>XSLServlet 1XSLServlet 2</td>
         </tr>
      </table>
   </body>
</html>
```

Nombre del archivo: *fecha.html*

```
<html xmlns:fecha="http://www.lauraponisio.com/xml/java/java.util.Date">
   <body>
      <p>Fri Nov 10 19:59:46 GMT-03:00 2000</p>
   </body>
</html>
```